



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**A METHOD FOR TESTING THE DYNAMIC ACCURACY
OF MICRO-ELECTRO-MECHANICAL SYSTEMS (MEMS)
MAGNETIC, ANGULAR RATE, AND GRAVITY (MARG)
SENSORS FOR INERTIAL NAVIGATION SYSTEMS (INS)
AND HUMAN MOTION TRACKING APPLICATIONS**

by

Jeremy L. Cookson

June 2010

Thesis Advisor:
Second Reader:

Xiaoping Yun
Marcello Romano

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2010	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: A Method for Testing the Dynamic Accuracy of Micro-Electro-Mechanical Systems (MEMS) Magnetic, Angular Rate, and Gravity (MARG) Sensors for Inertial Navigation Systems (INS) and Human Motion Tracking Applications			5. FUNDING NUMBERS	
6. AUTHOR(S) Jeremy Cookson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. . IRB Protocol number _____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <p>In this thesis a method for testing the dynamic accuracy of micro-electro-mechanical systems (MEMS) magnetic, angular rate, and gravity (MARG) sensors was developed. Many tests exist to check the static accuracy of MARG and inertial sensors or their individual components, but very few tests exist that adequately examine the dynamic accuracy of the final sensor package with sufficient precision and test repeatability.</p> <p>Based on a previous work that developed an inertial sensor test bench, a new test apparatus designed to model the motions of a human arm or leg was built using a rigid pendulum with MEMS MARG sensors attached on the end. Building materials were chosen to give minimal magnetic interference, and an optical encoder was used to accurately track the angle of the pendulum. A LabVIEW data acquisition system was built for data collection and a graphical user interface was written in MATLAB for easy data processing.</p> <p>The MicroStrain 3DM-GX1 and 3DM-GX3 sensors were tested on the new apparatus in a variety of dynamic motion tests, including free swinging in vertical and horizontal orientations, as well as "swing to impact" and semi-static tests, and their performances were compared for different target applications.</p>				
14. SUBJECT TERMS micro-electro-mechanical systems, MEMS, magnetic, angular rate, gravity sensor, MARG sensors, inertial navigation system, INS, inertial test, MicroStrain, 3DM-GX1, 3DM-GX3, CompactRIO, MATLAB GUI, dynamic accuracy test			15. NUMBER OF PAGES 217	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**A METHOD FOR TESTING THE DYNAMIC ACCURACY OF MICRO-
ELECTRO-MECHANICAL SYSTEMS (MEMS) MAGNETIC, ANGULAR RATE,
AND GRAVITY (MARG) SENSORS FOR INERTIAL NAVIGATION SYSTEMS
(INS) AND HUMAN MOTION TRACKING APPLICATIONS**

Jeremy L. Cookson
Civilian, Department of the Air Force
B.S., University of California, San Diego, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2010**

Author: Jeremy L. Cookson

Approved by: Xiaoping Yun
Thesis Advisor

Marcello Romano
Second Reader

R. Clark Robertson
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, a method for testing the dynamic accuracy of micro-electro-mechanical systems (MEMS) magnetic, angular rate, and gravity (MARG) sensors was developed. Many tests exist to check the static accuracy of MARG and inertial sensors or their individual components, but very few tests exist that adequately examine the dynamic accuracy of the final sensor package with sufficient precision and test repeatability.

Based on a previous work that developed an inertial sensor test bench, a new test apparatus designed to model the motions of a human arm or leg was built using a rigid pendulum with MEMS MARG sensors attached on the end. Building materials were chosen to give minimal magnetic interference, and an optical encoder was used to accurately track the angle of the pendulum. A LabVIEW data acquisition system was built for data collection and a graphical user interface was written in MATLAB for easy data processing.

The MicroStrain 3DM-GX1 and 3DM-GX3 sensors were tested on the new apparatus in a variety of dynamic motion tests, including free swinging in vertical and horizontal orientations, as well as “swing to impact” and semi-static tests, and their performances were compared for different target applications.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
B.	GOALS.....	3
C.	ORGANIZATION	4
II.	INTRODUCTION TO INERTIAL NAVIGATION SYSTEMS, SENSOR ORIENTATION, AND MEMS.....	7
A.	INERTIAL NAVIGATION AND REFERENCE FRAMES	7
1.	Reference Frames.....	8
a.	<i>Body Reference Frame</i>	<i>8</i>
b.	<i>Earth or Navigation Reference Frame</i>	<i>9</i>
2.	Position.....	10
3.	Orientation and Roll, Pitch, and Yaw	11
B.	MEMS TECHNOLOGY	12
C.	ACCELEROMETERS	13
D.	GYROSCOPES	14
E.	MAGNETOMETERS.....	15
F.	ADVANTAGES AND DISADVANTAGES OF MEMS INS AND MARG PACKAGES.....	16
G.	THE 3DM-GX1	17
H.	THE 3DM-GX3-25.....	20
I.	SUMMARY	22
III.	APPLICATIONS OF MEMS MARG SENSOR PACKAGES	23
A.	PERSONAL NAVIGATION	23
B.	IMMERSIVE VIRTUAL REALITY TRAINING	25
C.	COMMERCIAL APPLICATIONS	26
1.	Nintendo Wii.....	26
2.	Apple iPhone.....	27
D.	SUMMARY	27
IV.	TEST DESIGN AND APPARATUS	29
A.	SWINGING TESTS.....	29
B.	SUDDEN STOPS	30
C.	PREVIOUS TEST APPARATUS	30
D.	PROBLEMS WITH PREVIOUS TEST APPARATUS.....	31
E.	NEW APPARATUS DESIGN CONSIDERATIONS	32
1.	Simple but Sturdy	32
2.	Low-Cost Framework.....	33
3.	Low Magnetic Field	33
4.	Sudden Stop Capability	34
5.	Multiple Orientations	34
6.	Repeatability.....	34

F.	NEW APPARATUS DESIGN AND COST	34
1.	Apparatus Design With Impact Attachment.....	42
2.	New Apparatus Cost	43
G.	SUMMARY	44
V.	DATA COLLECTION	45
A.	ENCODER.....	45
B.	LABVIEW	48
C.	COMPACT RIO WITH FPGA	49
1.	NI 9403 Data Collection Device	51
2.	Encoder.VI.....	52
3.	Target.VI.....	57
a.	Loop 1	57
b.	Loop 2	61
4.	Host-to-Save.vi	66
5.	LabVIEW Data Collection Procedures.....	68
D.	ORIENTATION DATA CONVERSIONS	69
E.	ANGULAR VELOCITY AND ACCELERATION.....	70
1.	Theoretical Calculated Truth	70
2.	Actual Calculated Truth.....	71
3.	Calculated “Truth Data” Work-Arounds	78
F.	TIMING	79
1.	Data Collection Time-Stamping	80
2.	Correcting 3DM-GX1 and 3DM-GX3 Timing	80
G.	MATLAB DATA PROCESSING.....	83
1.	Turning Raw Data Into Results.....	83
2.	The Graphical User Interface “MEMS_Test”	84
H.	SUMMARY	88
VI.	TEST METHODOLOGY	89
A.	FREE SWINGING.....	89
B.	ARBITRARY SWINGING	91
C.	SEMI-STATIC: MOVE AND HOLD	92
D.	FREE SWING WITH IMPACT.....	94
E.	FREE SWING WITH IMPACT AND HOLD	96
F.	TEST MATRIX.....	98
G.	SUMMARY	100
VII.	RESULTS AND DATA PLOTS	101
A.	ACCURACY DEFINITIONS	101
B.	3DM-GX1.....	102
1.	Free Swinging.....	102
a.	Roll.....	102
b.	Pitch.....	104
c.	Yaw	105
d.	Free Swinging Tests’ Cross-Talk	107
2.	Arbitrary Swinging.....	109

	a.	<i>Roll</i>	109
	b.	<i>Pitch</i>	110
	c.	<i>Yaw</i>	111
3.		Semi-Static: Move and Hold	111
	a.	<i>Roll</i>	111
	b.	<i>Pitch</i>	113
	c.	<i>Yaw</i>	114
4.		Free Swing With Impact, and Free Swing With Impact and Hold	116
	a.	<i>Roll</i>	116
	b.	<i>Pitch</i>	117
	c.	<i>Yaw</i>	119
C.		3DM-GX3	120
	1.	Free Swinging	121
		a. <i>Roll</i>	121
		b. <i>Pitch</i>	122
		c. <i>Yaw</i>	124
		d. <i>Free Swinging Tests' Cross-Talk</i>	125
	2.	Arbitrary swinging	127
		a. <i>Roll</i>	127
		b. <i>Pitch</i>	128
		c. <i>Yaw</i>	129
	3.	Semi-Static: Move and Hold	129
		a. <i>Roll</i>	129
		b. <i>Pitch</i>	131
		c. <i>Yaw</i>	132
	4.	Free Swing With Impact, and Free Swing With Impact and Hold	134
		a. <i>Roll</i>	134
		b. <i>Pitch</i>	135
		c. <i>Yaw</i>	137
D.		OBSERVATIONS	138
E.		SUMMARY	140
VIII.		CONCLUSIONS	141
	A.	WHAT WAS ACCOMPLISHED	141
	B.	THE “BEST” SENSOR	141
		1. 3DM-GX1 Anomalies and Impact	142
		2. 3DM-GX3 Anomalies and Impact	142
		3. Overall Top Choice	143
	C.	RECOMMENDED FUTURE WORK	144
		APPENDIX A. LABVIEW DIAGRAMS	147
		APPENDIX B. MATLAB CODE	155
		A. MEMS_TEST.M, SELECTED FUNCTIONS	155
		B. READ_ENCODER_DATA_FILE.M	179

C.	ENCODER_TSPI.....	180
D.	PROCESS_TICK_TIME.M	182
E.	SENSOR_TIME_ALIGN.M.....	182
F.	ENCODER_FILTER.M.....	185
LIST OF REFERENCES.....		187
INITIAL DISTRIBUTION LIST		191

LIST OF FIGURES

Figure 1.	Body Coordinate Reference Frame.....	9
Figure 2.	Earth/Navigation Coordinate Reference Frame. After [7].....	10
Figure 3.	Body Coordinate Reference Frame Roll, Pitch, and Yaw.	11
Figure 4.	Earth/Navigation Coordinate Reference Frame Roll, Pitch, and Yaw.	12
Figure 5.	MEMS Tuning Fork Gyro Micromachining Process. From [8]	13
Figure 6.	2-Axis MEMS Accelerometer. From [10]	14
Figure 7.	1-Axis MEMS Gyroscope, Conceptual Structure. From [10]	15
Figure 8.	MEMS Magnetometer. From [11]	16
Figure 9.	MicroStrain 3DM-GX1, Top View.....	18
Figure 10.	MicroStrain 3DM-GX3, Top View.....	20
Figure 11.	Previous Test Apparatus Used in ENS Shaver's Thesis Work.....	31
Figure 12.	Front View Schematic of the New Test Apparatus.	35
Figure 13.	Top View Schematic of the New Test Apparatus.....	36
Figure 14.	Side View Schematic of the New Test Apparatus.	36
Figure 15.	Paper Model of Test Apparatus.	38
Figure 16.	1 inch to 3/8 inch Shaft Coupling.	39
Figure 17.	Completed Shaft to Encoder Coupling.	39
Figure 18.	Sensors on Mounting Board, Attached to Pendulum.....	40
Figure 19.	Final Test Apparatus, Horizontal Test Orientation.....	41
Figure 20.	Final Test Apparatus, Vertical Test Orientation.	41
Figure 21.	Top and Front View Drawings of Test Apparatus With Impact Board.....	42
Figure 22.	Test Apparatus Configured With Impact Board.	43
Figure 23.	A58 Encoder Timing Diagram. After [24]	47
Figure 24.	LabVIEW Real-Time Project Structure.....	50
Figure 25.	CompactRIO With NI 9403.....	51
Figure 26.	NI 9403 Connector Pin-Out to Encoder. After [25]	52
Figure 27.	FPGA-Direct Access to Encoder Data Pins.	53
Figure 28.	Encoder.VI Block Diagram.	54
Figure 29.	Encoder.VI Stacked Sequence, Broken Out.	56
Figure 30.	3DM-GX3 Target VI, Loop 1.....	59
Figure 31.	3DM-GX3 Target VI, Loop 1, True/False Frames Expanded.	60
Figure 32.	3DM-GX3 Target VI, Loop 2.....	62
Figure 33.	3DM-GX3 Target VI, Data Recombined for Euler Angles and Rates.	64
Figure 34.	3DM-GX1 Target VI, Recombined Stabilized Quaternion and Vectors.....	65
Figure 35.	3DM-GX3 Target VI, Loop 2, True/False Frames Expanded.	66
Figure 36.	Host-to-Save.vi.....	67
Figure 37.	Sample Encoder Data, Not a Smooth Curve.....	72
Figure 38.	FFT of Sample Encoder Data.	73
Figure 39.	FFT of Sample Encoder Data, - 10 Hz to 10 Hz.....	74
Figure 40.	FFT of Sample Encoder Data, ± 500 Hz, Zoomed to Show Noise.	75
Figure 41.	Sample Calculated Angular Velocity.....	76
Figure 42.	FFT of Sample Calculated Angular Velocity.	76

Figure 43.	FFT of Sample Calculated Angular Acceleration.....	77
Figure 44.	Sample Calculated Filtered Angular Velocity.	78
Figure 45.	Encoder and Sensor Data, Detail Showing Timing Offset.	81
Figure 46.	Error Plot With No Timing Correction.	82
Figure 47.	Error Plot With Timing Correction.	82
Figure 48.	MEMS_Test GUI, Startup Window.	85
Figure 49.	MEMS_Test GUI, Plotting Window Revealed.	87
Figure 50.	Sample Sensor Plots Without and With Adjustments, Respectively.....	88
Figure 51.	Free Swinging, Representative Encoder Angle, Low Release.	90
Figure 52.	Free Swinging, Representative Encoder Angle, Medium Release.	90
Figure 53.	Free Swinging, Representative Encoder Angle, High Release.	91
Figure 54.	Arbitrary Swinging, Representative Encoder Angle, Slow Movement.	92
Figure 55.	Arbitrary Swinging, Representative Encoder Angle, Fast Movement.	92
Figure 56.	Semi-Static, Representative Encoder Angle, Slow Movement.	93
Figure 57.	Semi-Static, Representative Encoder Angle, Fast Movement.	94
Figure 58.	Free Swing to Impact, Representative Encoder Angle, Low Release.	95
Figure 59.	Free Swing to Impact, Representative Encoder Angle, Medium Release.	95
Figure 60.	Free Swing to Impact, Representative Encoder Angle, High Release.....	96
Figure 61.	Impact and Hold, Representative Encoder Angle, Low Release.	97
Figure 62.	Impact and Hold, Representative Encoder Angle, Medium Release.....	97
Figure 63.	Impact and Hold, Representative Encoder Angle, High Release.	97
Figure 64.	3DM-GX1, Free Swinging Test, Roll Accuracy, Low Release Angle.....	103
Figure 65.	3DM-GX1, Free Swinging Test, Roll Accuracy, Medium Release Angle....	103
Figure 66.	3DM-GX1, Free Swinging Test, Roll Accuracy, High Release Angle.	104
Figure 67.	3DM-GX1, Free Swinging Test, Pitch Accuracy, Low Release Angle.....	104
Figure 68.	3DM-GX1, Free Swinging Test, Pitch Accuracy, Medium Release Angle. .	105
Figure 69.	3DM-GX1, Free Swinging Test, Pitch Accuracy, High Release Angle.....	105
Figure 70.	3DM-GX1, Free Swinging Test, Yaw Accuracy, Low Release Angle.	106
Figure 71.	3DM-GX1, Free Swinging Test, Yaw Accuracy, Medium Release Angle...	106
Figure 72.	3DM-GX1, Free Swinging Test, Yaw Accuracy, High Release Angle.....	107
Figure 73.	3DM-GX1, Free Swinging Test, Cross-Talk Plot, Roll Under Test.....	108
Figure 74.	3DM-GX1, Free Swinging Test, Cross-Talk Plot, Pitch Under Test.	108
Figure 75.	3DM-GX1, Free Swinging Test, Cross-Talk Plot, Yaw Under Test.	109
Figure 76.	3DM-GX1, Arbitrary Swinging Test, Roll Accuracy.....	110
Figure 77.	3DM-GX1, Arbitrary Swinging Test, Pitch Accuracy.	110
Figure 78.	3DM-GX1, Arbitrary Swinging Test, Yaw Accuracy.	111
Figure 79.	3DM-GX1, Semi-Static Test, Roll Accuracy.	112
Figure 80.	3DM-GX1, Semi-Static Test, Roll Accuracy, Zoomed in to Show Detail....	112
Figure 81.	3DM-GX1, Semi-Static Test, Pitch Accuracy.....	113
Figure 82.	3DM-GX1, Semi-Static Test, Pitch Accuracy, Zoomed in to Show Detail. .	114
Figure 83.	3DM-GX1, Semi-Static Test, Yaw Accuracy.....	115
Figure 84.	3DM-GX1, Semi-Static Test, Yaw Accuracy, Zoomed in to Show Detail. ..	115
Figure 85.	3DM-GX1, Impact Tests, Roll Accuracy, Low Release Angle.....	116
Figure 86.	3DM-GX1, Impact Tests, Roll Accuracy, Medium Release Angle.	117
Figure 87.	3DM-GX1, Impact Tests, Roll Accuracy, High Release Angle.	117

Figure 88.	3DM-GX1, Impact Tests, Pitch Accuracy, Low Release Angle.	118
Figure 89.	3DM-GX1, Impact Tests, Pitch Accuracy, Medium Release Angle.	118
Figure 90.	3DM-GX1, Impact Tests, Pitch Accuracy, High Release Angle.	119
Figure 91.	3DM-GX1, Impact Tests, Yaw Accuracy, Low Release Angle.	119
Figure 92.	3DM-GX1, Impact Tests, Yaw Accuracy, Medium Release Angle.	120
Figure 93.	3DM-GX1, Impact Tests, Yaw Accuracy, High Release Angle.	120
Figure 94.	3DM-GX3, Free Swinging Test, Roll Accuracy, Low Release Angle.	121
Figure 95.	3DM-GX3, Free Swinging Test, Roll Accuracy, Medium Release Angle.	122
Figure 96.	3DM-GX3, Free Swinging Test, Roll Accuracy, High Release Angle.	122
Figure 97.	3DM-GX3, Free Swinging Test, Pitch Accuracy, Low Release Angle.	123
Figure 98.	3DM-GX3, Free Swinging Test, Pitch Accuracy, Medium Release Angle. .	123
Figure 99.	3DM-GX3, Free Swinging Test, Pitch Accuracy, High Release Angle.	124
Figure 100.	3DM-GX3, Free Swinging Test, Yaw Accuracy, Low Release Angle.	124
Figure 101.	3DM-GX3, Free Swinging Test, Yaw Accuracy, Medium Release Angle. .	125
Figure 102.	3DM-GX3, Free Swinging Test, Yaw Accuracy, High Release Angle.	125
Figure 103.	3DM-GX3, Free Swinging Test, Cross-Talk Plot, Roll Under Test.	126
Figure 104.	3DM-GX3, Free Swinging Test, Cross-Talk Plot, Pitch Under Test.	126
Figure 105.	3DM-GX3, Free Swinging Test, Cross-Talk Plot, Yaw Under Test.	127
Figure 106.	3DM-GX3, Arbitrary Swinging Test, Roll Accuracy.	128
Figure 107.	3DM-GX3, Arbitrary Swinging Test, Pitch Accuracy.	128
Figure 108.	3DM-GX3, Arbitrary Swinging Test, Yaw Accuracy.	129
Figure 109.	3DM-GX3, Semi-Static Test, Roll Accuracy.	130
Figure 110.	3DM-GX3, Semi-Static Test, Roll Accuracy, Zoomed in to Show Detail.	130
Figure 111.	3DM-GX3, Semi-Static Test, Pitch Accuracy.	131
Figure 112.	3DM-GX3, Semi-Static Test, Pitch Accuracy, Zoomed in to Show Detail. .	132
Figure 113.	3DM-GX3, Semi-Static Test, Yaw Accuracy.	133
Figure 114.	3DM-GX3, Semi-Static Test, Yaw Accuracy, Zoomed in to Show Detail. .	133
Figure 115.	3DM-GX1, Impact Tests, Roll Accuracy, Low Release Angle.	134
Figure 116.	3DM-GX1, Impact Tests, Roll Accuracy, Medium Release Angle.	135
Figure 117.	3DM-GX1, Impact Tests, Roll Accuracy, High Release Angle.	135
Figure 118.	3DM-GX1, Impact Tests, Pitch Accuracy, Low Release Angle.	136
Figure 119.	3DM-GX1, Impact Tests, Pitch Accuracy, Medium Release Angle.	136
Figure 120.	3DM-GX1, Impact Tests, Pitch Accuracy, High Release Angle.	137
Figure 121.	3DM-GX3, Impact Tests, Yaw Accuracy, Low Release Angle.	137
Figure 122.	3DM-GX3, Impact Tests, Yaw Accuracy, Medium Release Angle.	138
Figure 123.	3DM-GX3, Impact Tests, Yaw Accuracy, High Release Angle.	138
Figure 124.	SubVI “Send 3DM-G Cmd.vi” Used in 3DM-GX1 Loop 2.	147
Figure 125.	SubVI “Get Serial Record.vi” Used in 3DM-GX1 Loop 2.	147
Figure 126.	SubVI “Decode 3DM-G Record.vi” Used in 3DM-GX1 Loop 2.	148
Figure 127.	SubVI “Send 3DM-GX3 Cmd.vi” Used in 3DM-GX3 Loop 2.	148
Figure 128.	SubVI “Get 3DM-GX3 Serial Record.vi” Used in 3DM-GX3 Loop 2.	149
Figure 129.	SubVI “Decode 3DM-GX3 Record.vi” Used in 3DM-GX3 Loop 2.	149
Figure 130.	LabVIEW VI for 3DM-GX1, Polled Mode, Loop 1.	150
Figure 131.	LabVIEW VI for 3DM-GX1, Polled Mode, Loop 2.	151
Figure 132.	LabVIEW VI for 3DM-GX1, Continuous Mode, Loop 2.	152

Figure 133.	3DM-GX3, Loop 2, Raw Sensor Measurements Selection.	153
Figure 134.	3DM-GX3, Loop 2, Raw Sensor Measurements and Orientation Matrix.	154

LIST OF TABLES

Table 1.	3DM-GX1 Specifications. From [13]	19
Table 2.	3DM-GX3 Specifications. From [15]	21
Table 3.	New Test Apparatus Parts Breakdown.	37
Table 4.	New Test Apparatus Cost Breakdown.	44
Table 5.	Encoder Pin-Out. From [24]	46
Table 6.	Non-Impact Test Matrix.	99
Table 7.	Impact Test Matrix.	100

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Over the past two decades, improvements in semi-conductor manufacturing techniques and the demand for low-cost miniaturized sensors has led to the development of micro-electromechanical systems (MEMS) sensors to replace larger, bulky sensors and open the door to future possibilities. MEMS accelerometers, gyroscopes, and magnetometers are being combined into a single package and used for diverse applications, from personal navigation systems to human motion tracking. When these three types of sensors are combined they are often referred to as magnetic, angular rate, and gravity (MARG) sensors. For personal navigation applications, a MEMS MARG sensor package can be used to track the distance traveled from an initial point without requiring any external references. In human motion tracking, MEMS MARG sensors can be placed at key locations on the body to track the wearer's motion, enabling technologies such as immersive virtual reality training for ground forces.

Although the MEMS MARG sensors are relatively cheap when compared to non-MEMS sensors, they come at the cost of decreased accuracy. In any application, the accuracy should be known, and both static and dynamic accuracies are important. Many tests exist to examine the static accuracy of MEMS MARG sensors. However, the test apparatuses that exist to measure the dynamic accuracy of accelerometers and gyroscopes are typically large metallic structures with robotic motors that have a significant impact on the magnetic field measured by a MARG, which could potentially add errors that are due entirely to the test apparatus. Additionally, the manufacturer specifications for dynamic accuracy are often said to be motion specific and, therefore, tests that exist for one type of motion may not adequately represent the expected performance of the sensor when subjected to another type of motion. The goal of this thesis was to develop a low cost and repeatable way to test the dynamic accuracy of MEMS MARG sensors for use in personal navigation or human motion tracking applications. Once a test apparatus was built, two sensors would be tested and compared to one another to verify the functionality of the test apparatus as well as to determine if one of the sensors was better suited for a personal navigation system or a human motion tracking application. The two sensors

tested were the MicroStrain 3DM-GX1 and the MicroStrain 3DM-GX3-25. These sensors were chosen because they were used in previous research and were readily available in the lab.

The test apparatus was designed to mimic certain motions that are expected in personal navigation and human motion tracking. Both of these applications have a significant amount of swinging, such as of the arms or legs and, therefore, the test apparatus was designed and built with a pendulum arm that swung freely and had the sensors mounted on the end. The test apparatus could be configured with the pendulum swinging vertically, or horizontally, with respect to the ground. An absolute optical encoder from Gurley Precision Instruments was attached to the axis of rotation of the pendulum and used to measure the angular displacement. Depending upon the orientation of the sensor at the bottom of the pendulum, the encoder could either provide roll, pitch, or yaw truth data. Theoretically, the encoder angular data also could be used to provide measurements of the angular rate and accelerations, but it was discovered that the encoder data were too noisy to provide sufficient accuracy as a truth source. Therefore, only the accuracies of the sensors' orientation output were examined.

The apparatus was made out of wood to minimize the impact on the local magnetic field, and the apparatus was made as rigidly as possible using 2 x 4s to minimize any out of plane motions that would not be captured by the encoder. To simulate walking motion, which involves a swing and a sudden stop, an impact arm was built with a shoe attached that could be mounted to the test apparatus and oriented appropriately to simulate the impact a sensor might experience in a personal navigation application.

Data collection was accomplished using a National Instruments CompactRIO cRIO-9012 that had a field-programmable gate array and was configured with an NI 9403 data collection module. Data collection virtual instruments were written in LabVIEW to read the encoder position and to get the orientation of the MicroStrain sensor. When collecting the data, it was extremely important that the timing match up between the encoder and the sensor, since any errors in the timing could be manifested as errors in the sensor accuracy, especially for sinusoidal motion. To mitigate timing errors, a common

clock reference was used on the CompactRIO and the data were further post processed to align the times in MATLAB. A graphical user interface was built in MATLAB to provide a simple way to reduce the data that were collected and then create standard plots to rapidly analyze the accuracy.

Once the new test apparatus and data collection equipment were functional, five different types of tests were run. For every type of test the roll, pitch, and yaw accuracies were tested, such that 15 unique test configurations were examined for each sensor. The five types of test were: Free swinging, arbitrary swinging, semi-static, free swing to impact, and free swing to impact and hold. In the free swinging tests, the pendulum arm was drawn back to a low, medium, or high mark to the right or left and then released and allowed to damp out to rest. All six conditions (low, medium, and high to the right and left) were run for each sensor. To conduct the arbitrary swinging tests the pendulum was moved about either slowly or quickly in a quasi-random fashion. The semi-static tests began at rest and then moved either slowly or quickly to the maximum deflection possible, about 70 degrees. The pendulum was held there for about five seconds before being released and allowed to swing to a stop.

For the impact tests, the apparatus was reconfigured with the impact arm. To conduct the free swing to impact test the pendulum was drawn back to the low, medium, or high mark and then released and allowed to bounce to a stop. The free swing to impact and hold test was conducted in a very similar manner, except that Velcro was revealed on the pendulum and impact arm and the pendulum was “caught” and held by the Velcro on impact.

The outcome of the testing showed that the new test apparatus worked well and could be used to collect repeatable dynamic accuracy data. The test results showed that that 3DM-GX1 met its specification requirements most of the time and outperformed the 3DM-GX3-25 for fast-motion tracking. However, the 3DM-GX1 had significant drift issues following dynamic motion that were revealed in the semi-static accuracy tests as well as the impact tests. By contrast, the 3DM-GX3-25 errors tended to go to zero over time following dynamic motions, but the output tended to lag the truth and the sensor had worse accuracy for fast dynamic motions than for slow motions. For both sensors the yaw

accuracy was the worst of the three axes. In addition, the impact testing revealed some accuracy discontinuities that both sensors experienced which would need to be dealt with for personal navigation applications.

Overall, a low-cost test apparatus was built and combined with an effective data collection system that could be used to test and analyze the dynamic accuracy of MEMS MARG sensors for use in personal navigation systems or human motion tracking applications. Due to the inaccuracies revealed in the dynamic tests, neither sensor tested would be an ideal solution for personal navigation or human motion tracking. Between the two sensors, the 3DM-GX1 would be the best for such applications since those applications have a lot of highly dynamic motions. However, it would be better to implement a combination of the two, since the 3DM-GX1 would be accurate over the short term and the 3DM-GX3-25 could be used to remove long-term drift errors.

LIST OF ACRONYMS AND ABBREVIATIONS

ARHS	Attitude Reference Heading Systems
FFT	Fast Fourier Transform
FPGS	Field Programmable Gate Array
GPS	Global Positioning System
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
MARG	Magnetic, Angular Rate, and Gravity
MEMS	Micro-Electro-Mechanical Systems
NED	North, East, Down
PC	Personal Computer
RMS	Root Mean Square
VI	Virtual Instrument
VR	Virtual Reality

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professor Yun for the opportunity to work on a topic that I found so interesting, as well as for the rapid reviews of my draft work and for steering me in the right direction when I ran into troubles.

Next I would like to thank James Calusdian. There is no way that this work would have been even half as thorough or successful without James. From the very beginning, he provided guidance and mentorship on everything from LabVIEW programming to the building of the test apparatus itself, and was a sounding board for me whenever I ran into trouble. I am sincerely grateful for all of his help.

I would like to thank my whole family for all of their love and support and for coming up to visit us during our crazy year in Monterey.

To my Jamie Jo, you are the most wonderful wife a nerd could ever have. Thank you for your unconditional love and support of me through this year. Thank you for your understanding of the long hours at school and for your constant encouragement for me to excel. And thank you for encouraging me to grow and trust in the Lord more every day. Prov. 3:5-6.

Finally, I want to thank the Lord God for giving me this opportunity. What a blessing to live and go to school in Monterey for a year. I pray that everything I do, including this thesis, would glorify Him.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Semiconductor manufacturing techniques have dramatically improved over the last two decades, and the ability to produce small, low-cost sensors such as accelerometers, gyroscopes, and magnetometers has become a reality. The widespread use of micro-electro-mechanical systems (MEMS) has permeated our daily lives, in everything from our cars to our phones. As the demand for MEMS sensors has increased, advances in the technology have made low-cost miniature Inertial Measurement Unit (IMU) and Magnetic, Angular Rate, and Gravity (MARG) sensor packages possible.

MEMS IMUs or MARG sensors can be used for many things, from personal navigation to immersive virtual reality training. When used in navigation or motion tracking the accuracy of the system is very important. The low price of MEMS IMUs and MARG sensors often comes at the cost of decreased accuracy and performance when compared to more traditional, and more expensive, IMUs and orientation sensor systems. Although the accuracy requirements of MEMS IMUs and MARG sensors are dependent upon the application of intended use, it is always important to understand the accuracy of the sensors being used.

There are publications that address the accuracy and calibration of individual inertial and magnetic sensors. The IEEE has published a standard set of recommended practices for testing inertial sensors that covers the types of tests to perform, the test equipment required, the data to collect, and how to analyze the data [1]. In addition, many technical papers also describe methods used when calibrating MARG type sensors and assessing their accuracy. In their work on the design and implementation of a new MARG sensor, Bachmann et al. describe the processes they used involving a Haas rotary tilt table and a PVC extension arm above the table to assess the accuracy and calibration of the sensor's gyroscopes and accelerometers [2]. In an evaluation of a 3D motion sensor with an integrated geomagnetic sensor, Chae and Park describe a magnetic field

generating apparatus that was used to artificially create a known magnetic field on the device under test for use in device calibration and assessing the accuracy of the magnetometers [3].

These methods and many of those in other publications tend either to require specialized expensive equipment or, alternatively, to address the accuracy as individual components rather than from a final system point of view, and then often in only static cases. For example, specialized equipment like a Haas rotary table can cost thousands or tens of thousands of dollars [4]. In addition, [1] suggests a number of structural and environmental constraints on test locations that all add to the cost of the facility needed to house such expensive pieces of equipment. At the other end of the spectrum, a simple and cheap way to test the accuracy of an accelerometer is to set it on a surface that is flat relative to the earth's gravitational field and see how closely it measures one times the acceleration due to gravity. This cheap and simple test, and others like it, focuses primarily on the static cases since precise dynamic motion is difficult to replicate without costly equipment and lab space.

Dynamic applications of MARG sensors have certain static and dynamic accuracy requirements. Often, sensors are employed such that, at certain times, the sensor is at rest—and therefore the static accuracy is very important—while at other times the sensor will be in dynamic motion, yet still expected to perform well through the full range of motion. Many times manufacturers base the performance of their MEMS IMU or MARG sensors primarily on their static performance, and little is elaborated with regard to the dynamic accuracy. Since motion in three dimensions over time can vary significantly, it is often hard to quantify with a simple specification number, and therefore rather than attempting to exhaustively test every possible dynamic motion in free space, which would be unrealistic, a blanket “dynamic accuracy” value is given with the caveat that it is motion dependent and may go outside those bounds. This value is typically a root mean square (RMS) error value calculated over the period of use of the system, and it does not address the likelihood of deviations during motion nor does it address the expected magnitude of any such deviations. Since the full MEMS sensor package needs to be tested before it is even selected for a given application so that the short-lived and

instantaneous dynamic motion errors can be known, a simple test for the dynamic accuracy of a MEMS MARG sensor or IMU is needed.

There are few examples of simple tests for dynamic accuracy of a MEMS MARG sensor. Since the sensors use magnetometers, many of the very precise and expensive tests using robotics do not provide representative results (the magnetic field from the metal robotic structure and the electric motors introduce errors that would not occur in the normal use of the sensor package), or those tests are cost prohibitive. In [5], Cutti et al. describe a simple test to determine the dynamic accuracy of an inertial sensor system. An array of sensors were mounted in the same orientation in a line on a piece of plastic and then manually rotated at a relatively constant rate. The rotation rate was maintained with the help of a metronome. Although very simple and cheap, the test is also very imprecise and does not produce repeatable results, since variations in how the board is spun mean that every sample is going to collect different motion. Also, there is no external reference to measure the ongoing accuracy and, as such, only an average value can be assessed. Many dynamic applications require accuracy through various rapidly changing motions, and so a simple mean value for the dynamic accuracy will not suffice. A repeatable method of accurately measuring the dynamic accuracy of a MEMS IMU or MARG sensor package is desired.

In his Master's thesis, ENS Shaver built a pendulum test apparatus to test the dynamic accuracy of a MARG sensor [6]. That apparatus was used as the starting point for the tests developed in this thesis.

B. GOALS

The objective of this thesis is to develop a low cost, repeatable way to test the dynamic accuracy of a MEMS MARG sensor, or IMU. In order to accomplish this, a suitable test apparatus must be built or modified such that repeatable tests can be performed. The apparatus must be able to reasonably replicate realistic dynamic motions that may be encountered in either personal navigation or an immersive virtual reality environment, since these are two of the target uses for the sensor packages. At least one independent source of truth data must be used. A data collection system must be built up

with an effort to synchronize the timing of all data collected, both the truth data and the data from the sensor under test. A simple way to analyze the results using a MATLAB graphical user interface is desired so that future testing is simplified. Finally, two different MEMS MARGs or IMUs were tested and their performances compared to determine which performs better for a selected application.

It is desired to keep the cost of the test apparatus as low as possible. The materials to build the test apparatus framework should cost around \$100. The cost of the data collection equipment and truth source may vary depending upon the level of precision desired. For this thesis, the goal was to use data collection equipment and a truth source that were readily available in the lab, not necessarily the cheapest solutions. If higher accuracy is required (or lower accuracy is acceptable), then it could be achieved by using a more (or less) expensive truth source and/or more (or less) expensive data collection equipment.

C. ORGANIZATION

This thesis is presented in the following chapters:

An introduction to the basic concepts of inertial navigation and spatial orientation is provided in Chapter II, and the components that make up a typical MARG or IMU are introduced. MEMS manufacturing technology is discussed with emphasis on the creation of MEMS MARGs and inertial navigation systems, and the Microstrain 3DM-GX1 and 3DM-GX3-25 sensors that were tested in this thesis are introduced.

In Chapter III, some of the existing and potential future uses of MEMS MARGs or IMUs are examined. Military, civilian first responder, and commercial consumer applications are all discussed.

The types of dynamic tests that were desired are introduced in Chapter IV, and the reasoning behind the tests and their applications to the real-world uses from the previous chapter are discussed. An existing test apparatus from [6] is examined for suitability to perform the required tests and found to be inadequate. Reasons for a new test apparatus are given and the design considerations of the new apparatus are covered.

In Chapter V, the specifics of how the “truth” data and sensor test data were collected and processed are presented. Detailed explanations are provided regarding what data were collected and also specifics are shown about the LabVIEW virtual instruments that were created to interface with the sensors and digital encoder. Finally, an explanation of how MATLAB was used to process the raw data into meaningful results is presented, and certain limitations that were encountered in the generation of truth data are addressed.

The test methodology and details about the test procedures are given in Chapter VI, as well as explanations regarding what data were collected, during which tests, for each of the sensors tested.

In Chapter VII, the specific test results for the 3DM-GX1 and the 3DM-GX3-25 are presented. The plots in this chapter are compared to the specifications introduced in Chapter II to identify whether or not the dynamic accuracy met the manufacturer’s specification claims for the desired motions. Where applicable, the sensors’ performances are compared.

Finally, the thesis is concluded in Chapter VIII by showing what was accomplished and by identifying areas of future work. The “best” sensor is selected for certain applications discussed in Chapter III based upon the test results presented in Chapter VII.

THIS PAGE INTENTIONALLY LEFT BLANK

II. INTRODUCTION TO INERTIAL NAVIGATION SYSTEMS, SENSOR ORIENTATION, AND MEMS

The basics of inertial navigation and orientation are introduced in this chapter to provide a framework and common reference for discussion throughout the thesis. MEMS manufacturing technology is briefly discussed, and the components that make up a MARG sensor are introduced at a very top level. Some of the pros and cons of using MEMS sensors for inertial navigation and orientation are discussed. Finally, the two sensors tested in this thesis are introduced, complete with specifications.

The combination of inertial data from orthogonally mounted accelerometers and orthogonally mounted gyroscopes to resolve position and orientation makes up an inertial navigation system (INS). With their inertial sensors and the ability to provide heading information and corrections via the magnetometer, MARG sensors can be used as miniature INSs. MARG sensor packages go by other names as well. MicroStrain refers to its MARG sensors as attitude reference heading systems (ARHSs). In order to understand what needs to be tested in these sensor packages, a basic understanding of the components that make one up is required. The building blocks used to make a MEMS MARG, INS, or ARHS are all the same types of sensors: accelerometers, gyroscopes, and magnetometers. The details of each component are not discussed in exhaustive detail, but the type of data gathered from each sensor is introduced and some background is given. For the sake of simplicity, this thesis treats INS, MARG sensors, and ARHS interchangeably and uses MARG or INS to identify the collection of accelerometers, gyroscopes, and magnetometers in a combined system.

A. INERTIAL NAVIGATION AND REFERENCE FRAMES

“Where am I?” This seemingly simple question can be incredibly challenging to answer. Over the years, methods for determining one’s location on the earth have been developed and improved upon. Using position fixing from an outside source, such as the stars, ancient seafarers could update their position on a featureless ocean. Alternatively, the method of deduced reckoning (also known as “dead” reckoning), was developed

wherein the navigator began at a known initial position and orientation and set off in a known direction, for a known period of time. Use of simple mathematics and a map or common reference frame makes it relatively easy to determine the new location. Pedometers, wheel counters, stars, sextants, compasses, and precise clocks have all been used as tools for determining position [7]. Inertial navigation is the modern extension of dead reckoning navigation techniques, only the distance traveled and orientation of the system are calculated using advanced inertial sensors like accelerometers and gyroscopes. Inertial navigation systems collect and process data that enable the position and orientation of the system to be calculated. MARG sensor packages act like an INS, but with an extra set of inputs from a magnetometer, which aids the inertial data processing.

1. Reference Frames

If the system is said to have moved “ahead” by two feet, what is the new location? Without more information, the answer is not determinable. In order for distances and orientations to have meaning, they must be interpreted within the context of a known reference frame. In a three-dimensional coordinate system, the reference frame establishes three axes about which the system is oriented and moves in relation to. For this thesis, two reference frames are important to understand, referred to as the “body” and “Earth/Navigation” reference frames.

a. Body Reference Frame

The body reference frame is defined with regard to the package that the system is contained within, and typically references the three orthogonal axes as x , y , and z . Often these axes are perpendicular to the package the sensors are in, or with respect to a larger function of the system (for example, a missile that fires forward may have a body reference frame such that the x -axis is aligned with the forward direction of flight, the z -axis is out the “top” of the missile, and the y -axis aligned in a right-hand rule fashion).

Both sensors tested in this thesis are in rectangular packages with the body reference coordinates clearly marked on the outside. Note that the body reference frame

may or may not exactly match the actual orientation of the sensors within the package. A typical body coordinate reference frame is shown in Figure 1.

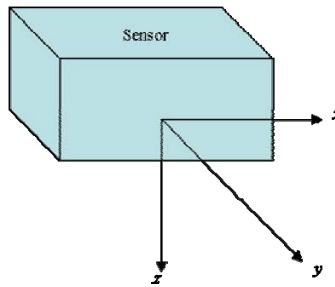


Figure 1. Body Coordinate Reference Frame.

b. Earth or Navigation Reference Frame

Most navigation and orientation that is of concern with the MEMS sensors under test in this thesis occurs on the earth, and therefore an “Earth” or “Navigation” reference frame is often used. In this system, a local level plane is established that is orthogonal to the gravitational vector, which points down toward the center of the earth (the “down” axis). On the local level plane, one vector points toward magnetic north and the other is aligned orthogonally, pointing east. It is important to emphasize that true north and magnetic north are two different directions. True north points toward the axis of rotation of the earth whereas magnetic north is aligned with the earth’s magnetic field. The Earth/Navigation reference frame is aligned with respect to magnetic north.

The north, east, down (NED) reference simplifies navigation over or near the surface of the earth, and an example is shown in Figure 2.

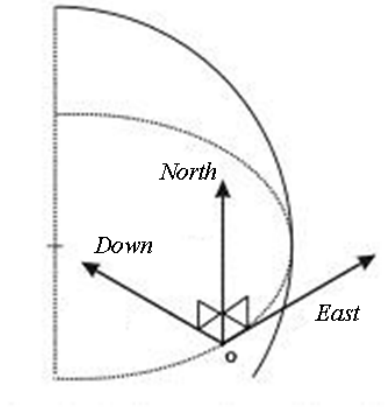


Figure 2. Earth/Navigation Coordinate Reference Frame. After [7]

2. Position

Given a known starting point, the position relative to that starting point may be calculated through a series of integrations. The accelerations in all three axes are measured and integrated over time to generate a velocity vector. That velocity vector may be integrated over time to determine the final position. For example, when starting at rest, a change in position of with respect to a single axis in the body reference frame may be calculated by

$$v_x(t) = \int_0^{\Delta t} a_x(t) dt \quad (1)$$

$$p_x(t) = \int_0^{\Delta t} v_x(t) dt \quad (2)$$

where $a_x(t)$ is the acceleration in the x direction over time, $v_x(t)$ is the velocity in the x direction over time, $p_x(t)$ is the position, and Δt is the time the object was moving.

When using this method to compute position, in order for it to be accurate, the acceleration must be known exactly. Any errors in the acceleration may propagate through to the velocity, and errors in the velocity will cause errors in position.

3. Orientation and Roll, Pitch, and Yaw

The orientation of the system is often referred to in the aeronautical terms of roll, pitch, and yaw. In the body reference frame, roll refers to a rotation about the x -axis, pitch refers to a rotation about the y -axis, and yaw refers to the rotation about the z -axis. The roll, pitch, and yaw in the body reference frame are shown in Figure 3.

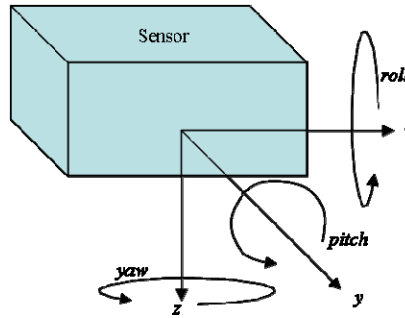


Figure 3. Body Coordinate Reference Frame Roll, Pitch, and Yaw.

In the earth/navigation reference frame, roll, pitch, and yaw are defined in a manner similar to that for the body reference frame with some slight but important differences. Roll is about the body x -axis, with zero degrees of roll defined when the body y -axis is parallel to the local reference plane. Pitch is a rotation about the y -axis or z -axis (depending upon the roll) and refers to the angle between the x -axis and the local reference plane, with zero degrees of pitch established when the x -axis is parallel to the local reference frame. Yaw, also referred to as heading in this reference frame, is a rotation about the “down” axis and is defined such that zero degrees of yaw/heading occurs when the x -axis is pointed along the north axis. The definitions of roll, pitch, and yaw in the earth/navigation frame are shown in Figure 4.

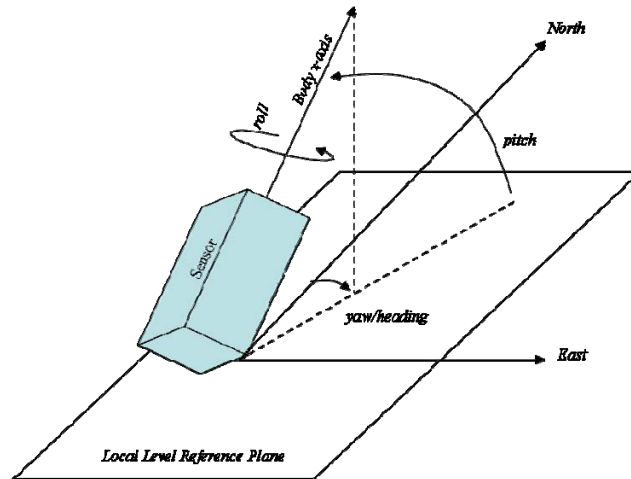


Figure 4. Earth/Navigation Coordinate Reference Frame Roll, Pitch, and Yaw.

B. MEMS TECHNOLOGY

Micro-electro-mechanical systems are physical machines with dimensions on the order of millimeters or less that are built using special techniques. The processes used to build MEMS sensors come primarily from the semiconductor industry [8]. Techniques such as depositing layers, masking, etching, electrostatic bonding, among others, are used to grow and build up small machines that physically are able move about in a measurable way and yet are fixed to the base structure. Electromagnetic forces are used to drive miniature “motors” that can vibrate masses. Springs and levers can be built and masses can be suspended from them. With improvements in semiconductor manufacturing techniques has come the ability to machine more and more complex MEMS.

In Figure 5, the steps to build a tuning fork gyroscope are shown merely to highlight the manufacturing process steps. The general technique to build any MEMS sensor involves many similar steps, though the exact details required to build modern MEMS sensors may be even more advanced and will certainly differ depending upon the application.

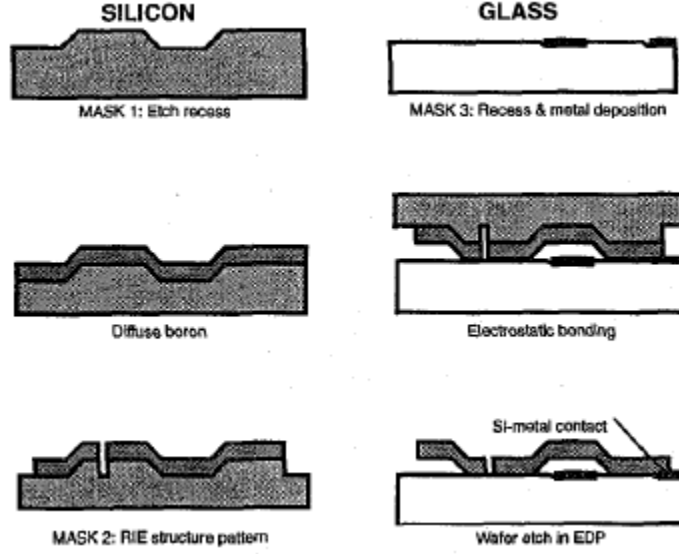


Figure 5. MEMS Tuning Fork Gyro Micromachining Process. From [8]

C. ACCELEROMETERS

Accelerometers are used to measure the acceleration applied to a particular axis. Early accelerometers suspended a mass in tension between two springs and measured the displacement of the mass with respect to the suspension structure to estimate the force being applied. Using Newton's Second Law of motion, we get

$$F = ma \quad (3)$$

where F is the applied force, m is the mass of the suspended proof mass, and a is the acceleration that can be estimated [9]. Acceleration is usually shown with units of m/sec^2 or "g", where $1g \approx 9.8\text{m/sec}^2$.

Although many other methods exist to measure acceleration using other physical phenomena, MEMS accelerometers are based on the observation of the same simple principles as early accelerometers, only on a much smaller scale. A proof mass is suspended between a set of springs and allowed to move freely in the direction of the axis under test. Electrodes that move with the mass are built in between electrodes that are fixed to the base structure and the capacitance is measured between the two. In this way,

the force may be estimated and Equation (3) solved to provide an estimate of the acceleration. A two-axis MEMS accelerometer is shown in Figure 6, with detail shown on how the electrodes measure the displacement.

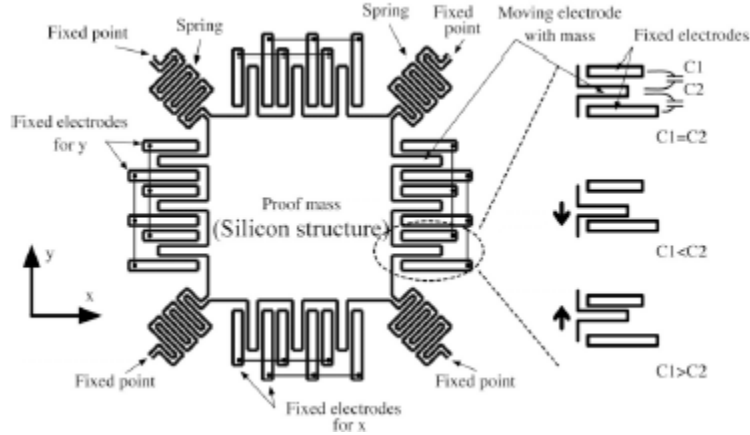


Figure 6. 2-Axis MEMS Accelerometer. From [10]

D. GYROSCOPES

Gyroscopes, also referred to as “gyros,” measure the angular rate turning about a particular axis. Many people are familiar with the classic children’s toy gyro that is spun and remains fixed in space while the structure is free to move about it. Early mechanical gyros operated in a similar fashion, and measured how fast the structure moved around a particular axis. Full-scale mechanical gyros, fiber optic gyros, ring laser gyros, and MEMS gyros all operate using different physical phenomena, but the end result is the same: gyros are used to gather angular rate data that are presented in rad/sec or deg/sec .

MEMS gyros typically operate using a vibrated mass and measure the forces on the structure caused by the Coriolis force as the gyro is rotated [10]. In Figure 7, a typical conceptual structure of a MEMS gyro is shown. With the inner structure vibrating, as the gyro is rotated about the z -axis the outer structure is moved in the x -direction proportional to the Coriolis force, and the displacement is measured using capacitive electrodes.

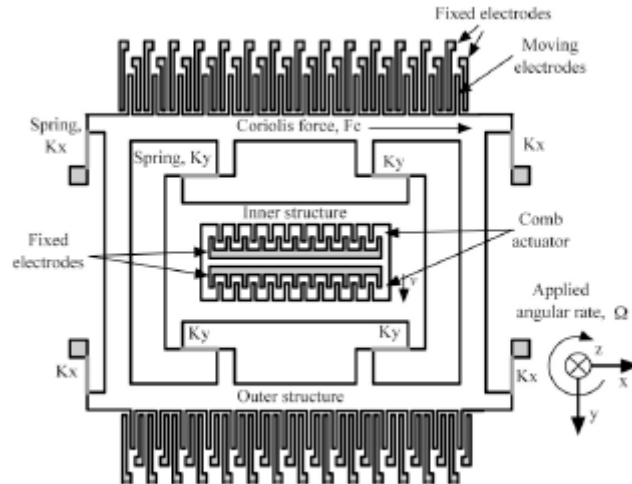


Figure 7. 1-Axis MEMS Gyroscope, Conceptual Structure. From [10]

E. MAGNETOMETERS

A magnetometer measures magnetic field vector in units of Gauss. There are many different ways to measure magnetic field, but all rely on the basic principles of electromagnetism. With regard to navigation, most people are familiar with a compass, which acts as a very crude magnetometer that gives direction only (no magnitude). Magnetic material floating in a liquid aligns itself with the earth's magnetic field and thus "points" to magnetic north. For navigation, a collection of MEMS magnetometers can be used to measure the magnetic vector and then resolve magnetic north based on the measured vector.

Certain methods to create MEMS magnetometers require the use of magnetic materials that are not commonly used in conventional semiconductor device fabrication and can be complicated to manufacture. Another way to measure the magnetic field that has been implemented on MEMS without the use of inherently magnetic material is to use the Lorentz force on a current carrying flexure in a magnetic field [11]. A current is passed through the central support beam, and then the flexure on the force beam caused by the presence of an out of plane magnetic field is measured using capacitive sensing plates. An example of this type of MEMS magnetometer is shown in Figure 8.

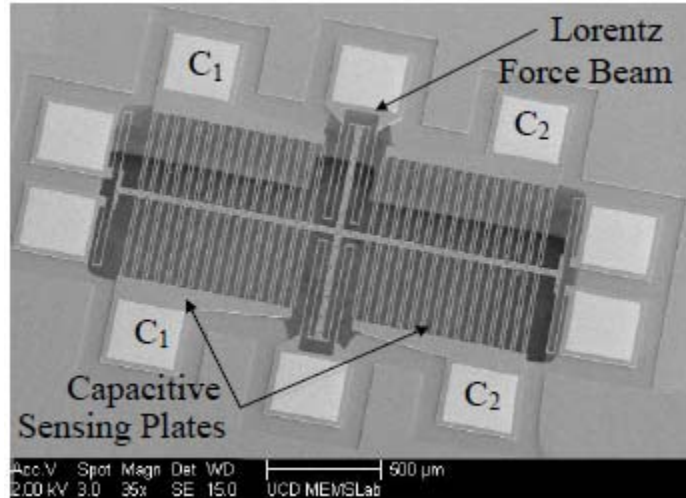


Figure 8. MEMS Magnetometer. From [11]

F. ADVANTAGES AND DISADVANTAGES OF MEMS INS AND MARG PACKAGES

The most obvious advantage to using a MEMS INS or MARG sensor package is size. With the dimensions of each sensor so small, the overall package required to contain triads of accelerometers, gyros, and magnetometers can still be relatively small, especially when compared to a traditional INS. This makes a MEMS INS or MARG package easily portable, able to fit in small unmanned aerial vehicles and unmanned submersible vehicles and even be wearable by a person in an unobtrusive manner, opening up a host of new possibilities for applications in inertial navigation and human motion tracking. Another major benefit of using MEMS technology is cost. With a single manufacturing process that is able to produce hundreds or thousands of sensors at a time, the economy of scale becomes significant, and the overall cost of each sensor decreases dramatically in comparison to the full size sensors.

The major concern with a MEMS INS or MARG sensor package is accuracy. The forces being measured are very small, and often the distances moved and changes in capacitance are very minute, thus the sensors are extremely susceptible to inaccuracies and drifting caused by noise, manufacturing defects, different thermal characteristics, and more. For example, a MEMS gyro experiences significant drift due to difficulty in

measuring absolutely no turn rate because of the way the structure is designed [10]. In another example, the integration of a MEMS accelerometer over time to produce velocity yielded a significant error [12]. Any use of a MEMS MARG sensor package needs to overcome these and other inherent inaccuracies in order to be used for the desired applications.

G. THE 3DM-GX1

The MicroStrain 3DM-GX1 is an AHRS that uses MEMS sensors to provide high fidelity orientation and inertial data via a serial port output. It was selected because it was used in previous thesis work and ongoing doctoral research, and therefore, it was available for immediate testing. The 3DM-GX1 combines three orthogonal accelerometers, three gyros, and three orthogonal magnetometers along with integrated onboard electronics to process the sensor data and provide a user requested output. The 3DM-GX1 is able to provide orientation data through a full 360 degrees of rotation in three orthogonal axes, has a 16 bit A-D converter, and utilizes a standard RS-232 serial data interface with variable baud rates to output selected data, including the quaternion, Euler Angles, attitude and heading, and more [13]. The 3DM-GX1 used for this thesis was in the housing provided by MicroStrain, and is shown in Figure 9.

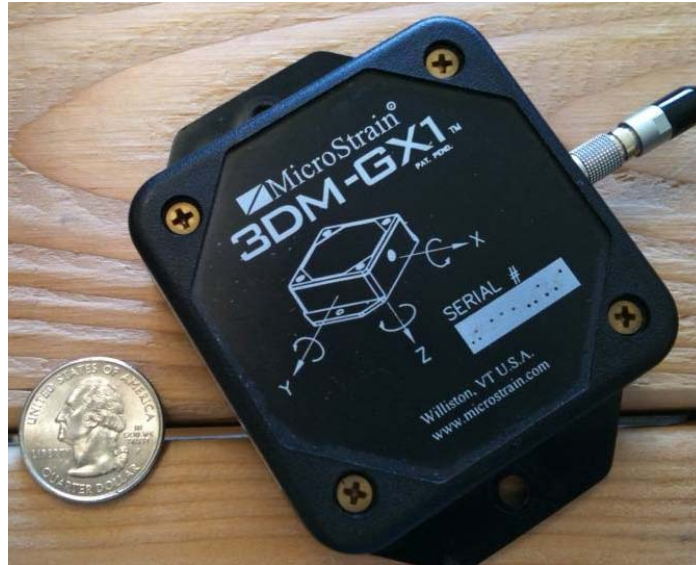


Figure 9. MicroStrain 3DM-GX1, Top View.

Utilization of the RS-232 serial port communication makes it possible to use either polled or continuous communication modes to interface with the 3DM-GX1. In polled communication mode, the 3DM-GX1 only transmits data when it is commanded to transmit. This requires the host to transmit a certain command, wait for the 3DM-GX1 to process it, and the 3DM-GX1 sends the output when it is finished processing. The alternate communication method is to use the continuous communications protocol. The desired command can be set and then the 3DM-GX1 continually outputs data to the serial port, updating the information in the data stream as soon as a new data set is processed. This requires the host computer to buffer large amounts of data since the same information is sent repeatedly. Both communication methods were attempted for this thesis and are discussed in more detail in Chapter V. The specifics about how to set polled or continuous modes and how to send commands and to translate the data, as well as the specific commands available, are all expanded in detail in [14].

The detailed specifications for the 3DM-GX1 are listed in Table 1. Note that the dynamic accuracy, specified by “cyclic” test conditions, was listed as ± 2 degrees, though nothing is elaborated on regarding specifics about what constituted “cyclic” motion.

Table 1. 3DM-GX1 Specifications. From [13]

Orientation range (pitch, roll, yaw)	360° all axes (orientation matrix, quaternion) ± 90°, ± 180°, ± 180° (Euler angles)
Sensor range	gyros: ± 300°/sec FS accelerometers: ± 5 g FS magnetometers: ± 1.2 Gauss FS
A/D resolution	16 bits
Accelerometer nonlinearity	0.20%
Accelerometer bias stability*	0.010 g
Gyro nonlinearity	0.20%
Gyro bias stability*	0.7°/sec
Magnetometer nonlinearity	0.40%
Magnetometer bias stability*	0.010 Gauss
Orientation resolution	<0.1° minimum
Repeatability	0.20°
Accuracy	± 0.5° typical for static test conditions ± 2.0° typical for dynamic (cyclic) test conditions & for arbitrary orientation angles
Output modes	matrix, quaternion, Euler angles, & nine scaled sensors with temperature
Digital outputs	serial RS-232 & RS-485 optional with software programming
Analog output option	4 channel, 0–5 volts full scale programmable analog outputs
Digital output rates	100 Hz for Euler, Matrix, Quaternion 350 Hz for nine orthogonal sensors only
Serial data rate	19.2/38.4/115.2 kbaud, software programmable
Supply voltage	5.2 Vdc minimum, 12 Vdc maximum
Supply current	65 mA
Connectors	one keyed LEMO, two for RS-485 option
Operating temp.	-40 to +70°C with enclosure -40 to +85°C without enclosure
Enclosure (w/tabs)	64 mm x 90 mm x 25 mm
Weight (grams)	75 grams with enclosure, 30 grams without enclosure
Shock limit	1000 g (unpowered), 500g (powered)
*Accuracy and stability specifications obtained over operating temperatures of -40 to 70°C with known sine and step inputs, including angular rates of ± 300° per second.	

H. THE 3DM-GX3-25

The MicroStrain 3DM-GX3-25, hereafter referred to as the 3DM-GX3, is the third generation ARHS from MicroStrain. It is smaller, lighter, and has faster processing speeds than the 3DM-GX1, though it provides essentially much of the same functionality, as well as some enhancements. It uses tri-axially mounted MEMS accelerometers, gyros, and magnetometers, as well as other internal sensors and a sensor fusion algorithm to provide orientation data in all three dimensions [15]. Similar data outputs were available as the 3DM-GX1, but the 3DM-GX3 did not provide the quaternion. Rather, the orientation matrix can be polled, or the orientation data can simply be output using the Euler angles. The 3DM-GX3 communicated via either RS-232 or USB. For this thesis, the RS-232 communication method was used. The 3DM-GX3 used for this thesis was in the enclosure provided by MicroStrain and is shown in Figure 10.



Figure 10. MicroStrain 3DM-GX3, Top View.

Similar to the 3DM-GX1, the 3DM-GX3 had a continuous mode of communication and a polled, or “active” mode of communication. Only the active mode was used for this thesis since the data rate achievable became limited by the data collection equipment and not the 3DM-GX3 sensor. In an improvement over the 3DM-GX1, the data output from the 3DM-GX3 were primarily in IEEE 754 floating point

(32 bit) and did not require much post processing to recover the orientation angles. The specifics about how to send commands, as well as which specific commands were available, are all expanded in detail in [16].

The detailed specifications for the 3DM-GX3 are listed in Table 2. Note that again the dynamic accuracy was specified by “cyclic” test conditions and cited as ± 2 degrees, though nothing is elaborated regarding specifics about what constituted “cyclic” motion. Also note that although there appear to have been improvements in the accuracy and stability of the sensors, the overall static and dynamic accuracy specifications were not improved over the 3DM-GX1.

Table 2. 3DM-GX3 Specifications. From [15]

Orientation range	360° about all axes
Accelerometer range	± 5 g standard ± 2 g, ± 18 g, and ± 50 g also available
Accelerometer bias stability	± 0.005 g for ± 5 g range ± 0.003 g for ± 2 g range ± 0.010 g for ± 18 g range ± 0.050 g for ± 50 g range
Accelerometer nonlinearity	0.20%
Gyro range	$\pm 300^\circ/\text{sec}$ standard, $\pm 1200^\circ/\text{sec}$, $\pm 600^\circ/\text{sec}$, $\pm 150^\circ/\text{sec}$, $\pm 75^\circ/\text{sec}$ also available
Gyro bias stability	$\pm 0.2^\circ/\text{sec}$ for $\pm 300^\circ/\text{sec}$
Gyro nonlinearity	0.20%
Magnetometer range	± 2.5 Gauss
Magnetometer nonlinearity	0.40%
Magnetometer bias stability	0.01 Gauss
A/D resolution	16 bits (SAR) (oversampled to 17 bits)
Orientation Accuracy	$\pm 0.5^\circ$ typical for static test conditions $\pm 2.0^\circ$ typical for dynamic (cyclic) test conditions & for arbitrary orientation angles
Orientation resolution	$<0.1^\circ$
Repeatability	0.2°
Output modes	acceleration, angular rate and magnetic field deltaAngle and deltaVelocity Euler angles rotation matrix
Interface options standard:	USB 2.0 or RS232 OEM: USB 2.0 / TTL serial (3.3 volts)
Data rate	1 Hz to 1,000 Hz
Filtering	sensors sampled at 30 kHz, digitally filtered

	(user adjustable) and scaled into physical units; coning and sculling integrals computed at 1 kHz.
Baud rate	115,200 baud to 921,600 baud
Supply voltage	standard: 4.4 to 6 volts [up to 15 volts operation possible at limited temp range or low duty cycle] OEM: 3.2 to 5.5 volts
Power consumption	80 mA @ 5 volts with USB
Connectors	micro-DB9, OEM: Samtec FTSH-105-01-F-D-K
Operating temp.	-40 °C to +75 °C (consult factory for higher temperature operation)
Dimensions	44 mm x 25 mm x 11 mm - excluding mounting tabs, width across tabs 37 mm, OEM: 38 mm x 24 mm x 12 mm
Weight	18 grams RS-232 and USB, 11.5 grams OEM
Shock limit	1000 g (unpowered), 500g (powered)
*Accuracy and stability specifications obtained over operating temperatures of -40 to 70°C with known sine and step inputs, including angular rates of $\pm 300^\circ$ per second.	

I. SUMMARY

The basics of inertial navigation and body orientation were discussed in this chapter, and the building blocks of MEMS MARG sensors were introduced, providing a quick overview of the type of data that each sensor can measure and briefly explaining how the sensor measures it. The two reference frames used in this thesis were introduced, and details about the pros and cons of MEMS sensors were provided. Finally, the specific sensors that would be tested in this thesis, the MicroStrain 3DM-GX1 and 3DM-GX3, were introduced and the manufacturer's specifications were listed.

In the next chapter, some of the potential applications of MEMS MARG sensor technology are introduced with emphasis on personal navigation and human motion tracking. In addition, current commercial applications of MEMS inertial and magnetic sensors are also mentioned to familiarize the reader with the capabilities of MEMS sensors.

III. APPLICATIONS OF MEMS MARG SENSOR PACKAGES

In this chapter, a few of the existing and potential future applications of MEMS MARG sensor packages are introduced. Emphasis is given to military and first responder applications, though some commercial applications are also given to show where MEMS sensors of this sort are in use that may be familiar to the reader.

A. PERSONAL NAVIGATION

The concept of having a system that tracks a person's location and makes that location readily known to the user has been around for a number of years. The establishment of the USAF Global Positioning System (GPS) satellite constellation in 1993 enabled military and civilian users alike to be able to instantly and accurately track their locations with varying levels of accuracy [7]. The basic functionality works by receiving a signal from a minimum of four different satellites and then resolving the receiver position by solving four equations simultaneously for x , y , and z position and a clock delta parameter. Different satellites in view may be used to solve for the user's position with differing levels of accuracy.

When access to the GPS signal is denied, either due to enemy jamming or due to masking the receiver (as when going indoors), the ability to use GPS to navigate is lost. Alternative solutions exist for navigating indoors, but many require the use of external equipment or a-priori knowledge of the indoor environment via a map or other means. A cleaner solution that requires no external infrastructure would be to have a personal navigation system attached to the body to track location.

In their work on self-contained position tracking of human movement using small inertial/magnetic sensor modules, Xiaoping Yun et al. showed that it was possible to use a MARG sensor package to accurately track a human position through walking, running, and side stepping [12]. Inherent errors of the MEMS sensors were removed through clever filtering and correction of the sensors' drift using zero foot velocity updates.

In her thesis, “A position tracking system using MARG sensors,” Miryung Um [17] built on previous work to develop a position tracking system using MARG sensors. A MARG was firmly attached to the top part of a shoe near the ankle, and using knowledge of the human gait cycle and an algorithm that zeroed out velocity errors when the foot was not actively swinging, a rudimentary personal INS was developed. The work demonstrated the ability to track position with reasonable accuracy while walking a straight line and climbing stairs.

There are a number of potential military applications for personal navigation. When combined with GPS receiver, a personal INS would add a layer of protection against jamming or spoofing of the GPS. The INS data could be combined with the GPS data using a Kalman filter, as many aircraft systems do, providing a reliable and continuous means of navigation with and without GPS. It would also provide an independent source that may alert the user to the presence of attempted spoofing or other electronic attack. Such an integrated system would provide the ability to navigate electronically in areas where GPS reception is poor, such as in canyons or inside buildings. Further, this personal data could be integrated with a transmitter allowing a battlefield commander to know the exact location of all troops, even in an urban environment when troops are scattered through buildings.

The civilian first responder applications are similar to military applications. A firefighter with a personal navigation system could have a map of a building with his or her current location displayed on it, allowing rapid movement through large, complex smoke-filled buildings. Such a map and display could even be integrated into a helmet mounted display, further enhancing its utility. Also, like the military application, a personal INS could be combined with a transmitter so that those in command of a large scale rescue event, such as that on September 11, 2001, would know precisely where all of the rescue personnel were at any given time. This would enable more efficient and safer direction of resources.

B. IMMERSIVE VIRTUAL REALITY TRAINING

The United States Military is one of the best-trained militaries on the planet, but that training comes at a cost. In FY09 alone, nearly \$1.1 billion was allocated for active duty Army and Marine Corps recruit and specialized skill training [18]. Although the Air Force also spends a significant amount on training, the use of simulators has provided a way to cut the cost of training pilots. The A-10, F-22, and F-35 all are single-seat fighters with no dual-seat instructional flying variants. Prior flight training and simulators are enough to sufficiently bring pilots up to speed so that their first flight in one of these aircraft is a solo flight, but training in a cockpit is different than combat training on the ground.

Many different simulators have been developed in an attempt to address training troops on the ground. They range from the very simple, essentially video games that run on laptop computers [19], to sophisticated systems using virtual reality goggles and omni-directional treadmills. These systems may provide realistic looking environments, but they do not provide realistic training. One reason flight simulators work for pilots is that the pilots are able to manipulate the same controls and interact with the environment in the same way as they would in a real aircraft. Current simulations for troops on the ground do not provide this fidelity of real-world motion combined with simulation.

One solution is immersive virtual reality (VR) training. In immersive VR, a user can digitally enter a virtual world through the use of a portable viewer and an integrated motion tracking system that is able to track the user and “place” them in the simulation. To do this requires accurate human motion tracking. Motion capture technology popular in the film industry relies on external cameras and wearable reference nodes, which requires a significant infrastructure. In [20], Eric Bachmann demonstrated the feasibility of creating a “source-less” human motion tracking system using MEMS MARG sensors placed at key locations on the body. A filtering algorithm was developed, and the ability to manipulate and control a virtual human body model in near real time was shown. The self-contained sensor systems have an advantage over those that require external cameras or sensors to track since external cameras or sensors could be masked by common user

motions. In addition, they rely on extra infrastructure and are limited to a confined space. Utilization of MEMS MARG sensors on the hands, feet, shoulders, helmet, rifle, and other locations allows position and orientation data to be fed into a portable central processor running the simulation, and conceivably there would be no spatial or masking limitations. Those in training could go through many of the same motions they would in real life, enhancing the utility and lowering the cost of training. Use of this technology could make it possible to create virtual environments, and have troops “walk the streets” of a city or town in which they were going to be conducting operations, prior to even leaving their home base.

C. COMMERCIAL APPLICATIONS

MEMS sensors are working their way into the mainstream, and many popular consumer electronics utilize some or all of the MEMS sensors that make up MARG packages and MEMS INSs. These applications are presented here to provide the reader with familiar and tangible references regarding what the sensors that make up a MEMS MARG are capable of doing.

1. Nintendo Wii

First introduced in 2006, the Nintendo Wii dramatically changed the way users interact with a gaming system. The use of a remote that detected the motion of the users enabled a new level of interactive gaming. Players used a remote with normal video game buttons but also with a set of sensors that translated the users’ motions into virtual actions.

The Wii remote contains a three-axis MEMS accelerometer built by Analog Devices [21]. These sensors allow the remote to detect movement and identify motions like swinging, while the game software translates these actions into on-screen game play. Games like tennis, golf, bowling, fishing and more are controlled by moving the remote in a similar fashion as the real-world activities.

In 2009, Nintendo released the Wii MotionPlus attachment, which Nintendo advertised “brings every twist of the wrist or turn of the body to life” [22]. The additional

sensor pack integrated three MEMS gyroscopes, which enabled the remote to more accurately determine its orientation. This translated into more precision when replicating complex natural motions.

With tri-axial accelerometers and gyros, the Wii remote with MotionPlus is a state-of-the-art commercial use of MEMS sensor technology.

2. Apple iPhone

Another revolutionary piece of technology is Apple's iPhone, originally released in 2007. The iPhone pioneered or made popular many new features of so called "smart phones," and one of the subtle enabling technologies used in those new features was the integration of a three-axis MEMS accelerometer.

Most of the uses of the accelerometers in the iPhone rely on the acceleration due to the earth's gravity. Programs and applications use the three-dimensional gravity vector detected by the sensors to identify the orientation of the iPhone. This enables switching between portrait and landscape mode and also provides an input for many third-party developed games and applications, such as a remake of the children's game "Labyrinth" or an electronic carpentry level. Newer iPhone software also enabled a "shake" input, where the accelerometers detect a certain threshold of shaking in a period of time, which can be used to shuffle music when using the music features, among many other uses [23].

The latest version of the iPhone, the 3Gs, introduced a magnetometer as another embedded sensor. A digital compass application allows users to find magnetic north and their current heading (yaw in the earth reference frame). Both the magnetometer and the accelerometers function in the background of the iPhone, yet they are important and integral parts to the technology.

D. SUMMARY

In this chapter, some of the current applications of MEMS MARG sensor technology in both personal navigation and human motion tracking were presented.

Potential applications for future military and civilian use were introduced. Finally, some common commercial uses of MEMS sensors were presented to connect the reader with the capability of MEMS sensors.

In the next chapter, the testing to be accomplished in this thesis is developed and the test apparatus that was designed, and ultimately built, is presented. The chapter begins with a discussion about the types of dynamic tests to be simulated then a previous test apparatus is examined before the requirements and specifications for the final new test apparatus are introduced.

IV. TEST DESIGN AND APPARATUS

The types of tests that will be accomplished are discussed in this chapter. The rationales for the tests are provided and a previous test apparatus is examined for suitability. New apparatus design considerations are put forward and the final design is presented.

A. SWINGING TESTS

Early implementations of personal navigation systems, such as [12] and [17], have placed the MEMS MARG sensor package on the foot near the ankle. The accelerations of the foot were measured throughout each step and were used to calculate the distance travelled. The motion of walking is described by the human gait cycle, which has a stance phase and a swing phase. In the swing phase, the leg is moving and swinging through the air to the next position. In the stance phase, the foot is stationary while the other foot is moving. In the referenced examples, the sensor package attached to the foot moved during the swing phase and, therefore, the dynamic motion that needs to be tested for personal navigation applications should mimic the leg swinging.

In human motion tracking applications for immersive virtual reality, as in [20], sensors may be placed on the wrists, elbows, shoulders, head, and elsewhere to track the orientation of the individual. When moving about, the sensors can expect to experience some periodic motion, as when walking, but must also expect to experience somewhat random motions since a person's orientation is not fully predictable. The ability to have a quasi-random motion test is needed in order to test appropriately for human motions.

In very simple terms, many of the dynamic motions, both in personal navigation and human motion tracking for immersive VR training, may be approximated by swinging around a pivot point. When walking, the knee acts as a pivot point, and the foot with the sensor attached swings freely below. When pointing an arm or a rifle, the elbow or shoulder act as the pivot and the arm or rifle may swing horizontally about that point. These descriptions do not cover the two- or three-dimensional translation that also occurs

in these motions, but to a first-order approximation, walking or swinging an arm may be thought of as a pendulum swinging about a point.

Since swinging about a fixed point mimics the dynamic motions found in personal navigation and immersive VR training, swinging tests were the primary focus of this thesis.

B. SUDDEN STOPS

Walking is a continuous set of transitions from swinging to not moving and then back to swinging again. The performance of the MEMS INS is important through all phases. In [12], the velocity was zeroed out once it was identified that the foot was no longer moving. Therefore, the accuracy and speed of the transition from moving to being stopped is very important.

In addition to testing the sensors in a freely swinging motion, tests were developed so that the MEMS INS was swung once and then came to a sudden stop. The speed at which the sensor was stopped was intended to mimic that of a foot hitting the ground and transitioning from the swing phase to the stance phase of the gait cycle.

C. PREVIOUS TEST APPARATUS

In [6], ENS Shaver built a simple test apparatus that was a pendulum with the sensor unit attached to one end of a wooden dowel and the pivot point connected to an optical absolute encoder. A picture of the apparatus is shown in Figure 11.



Figure 11. Previous Test Apparatus Used in ENS Shaver's Thesis Work.

The apparatus shown in Figure 11 is able to test the sensors in various orientations through a repeatable swinging motion. The length of the pendulum is adjustable, which allows different moment arms to be tested simulating shorter or longer arms or legs. Test setup is quick, and overall the cost of the test apparatus was low, with the exception of the encoder.

D. PROBLEMS WITH PREVIOUS TEST APPARATUS

Although it worked for initial tests, there were many things that made the old test apparatus unsuitable for the testing in this thesis. The major problem was rigidity. The old apparatus' pendulum shaft was a 1/4 inch wooden dowel that was attached to the encoder using a series of bearings and a k-coupling. There was significant lateral translation in the system, especially when the pendulum was lengthened. The pendulum was susceptible to wobbling slightly, a motion that was not captured by the encoder. Since the desire was to have repeatable motion that was instrumented by an outside

source, this wobbling was unacceptable. Further, the lack of rigidity posed more problems when the apparatus was attempted to be operated with the pendulum horizontal to the ground, as the pendulum bowed significantly.

Another problem with the previous apparatus was developing a way for it to stop quickly and precisely. Manually catching the pendulum identified problems that would be encountered if the pendulum were suddenly stopped. Since there was so much free-play in the system, when the pendulum was stopped suddenly, the entire pendulum and connection unit tended to wobble and shake while dissipating the energy. None of these motions were accurately captured by the single encoder.

The other problems encountered were minor, but needed to be addressed. There was no way to start tests in the same place each time, making it difficult to obtain repeatable results. Also, the existing unit was not designed to work well with the pendulum horizontal to the ground. Finally, there was no way to test two sensors on the same test apparatus.

E. NEW APPARATUS DESIGN CONSIDERATIONS

Given that the existing apparatus would not work, a new test apparatus was designed, using the old apparatus as a starting point for the general concept but attempting to address the problems encountered as well as enhance its utility. The following considerations were included in the design.

1. Simple but Sturdy

Since a lack of rigidity was one of the major problems with the old apparatus, the new apparatus was required to be sturdy. The frame needed to be heavy and not susceptible to shaking, and the pendulum needed to be solid and not prone to flexion. The shaft that the pendulum would rotate about needed to be thicker and attached rigidly using mount bearings to avoid the wobbling problems encountered with the old apparatus.

Although a sturdy construction was required, the apparatus needed to be simple to build, without any special tools. A few common power saws and a simple power drill were the only “special” tools allowed. Additionally, the method of construction needed to be basic, with simple joints and connections rather than using fancy carpentry to fasten the unit together.

2. Low-Cost Framework

An effort was made to keep the construction of the test apparatus as cheap as possible without sacrificing utility. Simple construction materials that could be found at any hardware store and lumberyard were used as much as possible. Cheaper pressed steel flange bearings were chosen, as opposed to stainless steel bearings, because they provided the same function at fraction of the cost. The low cost (near \$100) makes it easier for others on a tight budget to build a similar test apparatus if so desired. Note that the encoder had already been purchased, and it was not included as part of the cost of the test apparatus.

3. Low Magnetic Field

Since the MARG sensors have a magnetometer that plays a crucial role when translating the body reference frame to the navigation reference frame, it was important that the test apparatus not significantly distort or change the magnetic field measured by the sensors. The test apparatus would be operated in an indoor environment similar to that expected that a personal INS would operate in, with metal tables nearby and computers running. However, since in practice a personal INS could be made and attached to the body without adding significant extra magnetic field, it was important to minimize the magnetic signature of the test apparatus as much as possible.

Since a metal structure was ruled out, both wood and PVC plastic were examined as potentially suitable solutions. However, PVC did not meet the rigidity requirements discussed above and, therefore, a wooden structure was chosen. Also, in selecting the bearings, an inexpensive option was to use cast iron based flange bearings. However, these were not selected because of the potential magnetic field change caused by the cast

iron. Instead, pressed steel were selected since the cost increase was minimal and the difference in magnetic interference potentially significant.

4. Sudden Stop Capability

The new apparatus was designed for a free-swinging motion, similar to the old apparatus. However, it was desired to have the capability to add an attachment that brought the pendulum to a rapid and predictable sudden stop after one cycle of a swing, similar to the swing-to-stance phase of the gait cycle. The attachment needed to be firm enough to bring the pendulum to a stop without too much wobbling, yet temporary so that it could be added and removed quickly and easily, depending upon the testing desired.

5. Multiple Orientations

Since mimicking the walking motion required the pendulum to be oriented vertically, but mimicking a pointing arm motion required the pendulum to be oriented horizontally, the ability to easily change the orientation of the pendulum, rather than developing two separate test apparatuses, was required.

6. Repeatability

The previous test apparatus provided no way to ensure that any two tests started or finished at approximately the same angle. Although the angle of the pendulum would be measured by the encoder, it was desired to have an independent way to ensure tests could easily be conducted in a repeatable manner.

F. NEW APPARATUS DESIGN AND COST

The final test apparatus was designed to be built primarily out of standard 2 x 4 inch boards. The base would be 1/4 inch Masonite board, that did not serve structural purposes as much as it was something common for the 2 x 4s to be attached to. The pendulum would be a 2 x 4, and it could be changed out with varying lengths of 2 x 4s if

The schematics of the new test apparatus from the front, top, and side views are shown in Figures 12, 13, and 14, respectively. The drawings are not to scale, but the exact dimensions corresponding to the pieces labeled in the three views are found in Table 3.



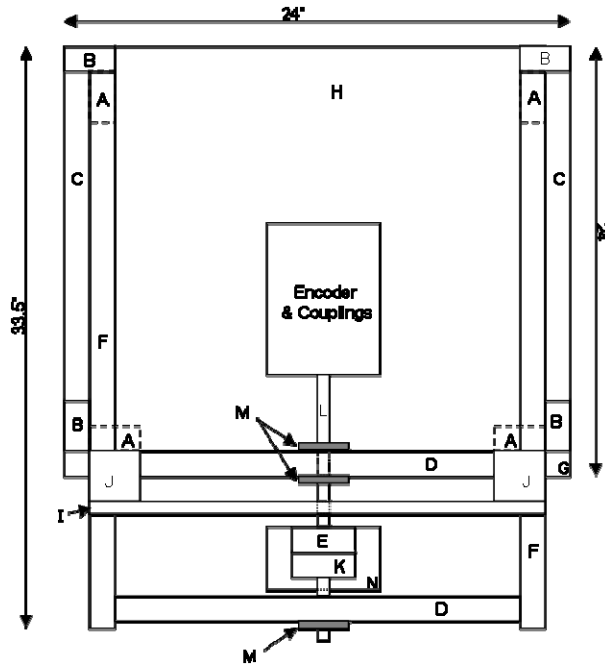


Figure 13. Top View Schematic of the New Test Apparatus.

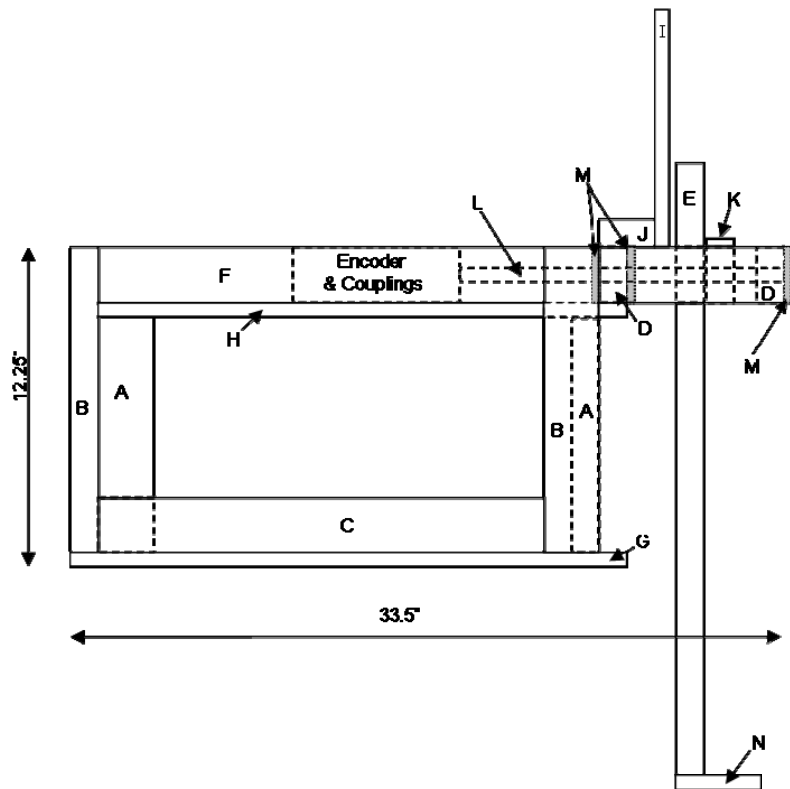


Figure 14. Side View Schematic of the New Test Apparatus.

The parts needed to build the new test apparatus are contained in Table 3. Note that these are already the “cut” pieces, not necessarily the parts that were purchased. Also, these are the actual dimensions used (a standard 2 x 4 actually measures 1.75 in x 3.5 in).

Table 3. New Test Apparatus Parts Breakdown.

Use	Dimensions	Quantity	Label in 3-Views
Base legs for upper table	1.75 in x 3.5 in x 8 in	4	A
Outer legs	1.75 in x 3.5 in x 11.75 in	4	B
Bottom flat vertical brace	1.75 in x 3.5 in x 17.25 in	2	C
Cross braces	1.75 in x 3.5 in x 18 in	2	D
Pendulum	1.75 in x 3.5 in x 24 in	1	E
Cantilevers	1.75 in x 3.5 in x 32 in	2	F
Base	0.25 in x 24 in x 24 in	1	G
Top	0.25 in x 21 in x 22.5 in	1	H
Angle Reference Board	0.25 in x 21 in x 12 in	1	I
Reference Board Mounts	1.75 in x 3.5 in x 3.5 in	2	J
Shaft/Pendulum Joining Piece	1.75 in x 3.5 in x 4 in	1	K
1 inch Wooden Dowell	1 in x 18 in	1	L
Flange Mount Bearings	1 in inner diameter	3	M
Sensor Mounting Board	0.25 in x 6 in x 7 in	1	N
K-Coupling	0.375 in	1	--
K-Coupling	0.25 in	1	--
PVC Pipe Cap	3/4 in schedule 40	1	--
Bolt	0.375 in	1	--
Woodscrews	3 in	1 box	--
Woodscrews	1.625 in	1 box	--
Epoxy	--	1 bottle	--
Wood glue	--	1 bottle	--

A scale paper-model mockup was built to identify problems early, before parts were already cut, and to show exactly how everything would fit together. The model

helped identify additional areas that needed structural support and provided an easy template to work from when building the final product. A picture of the scale model is shown in Figure 15.

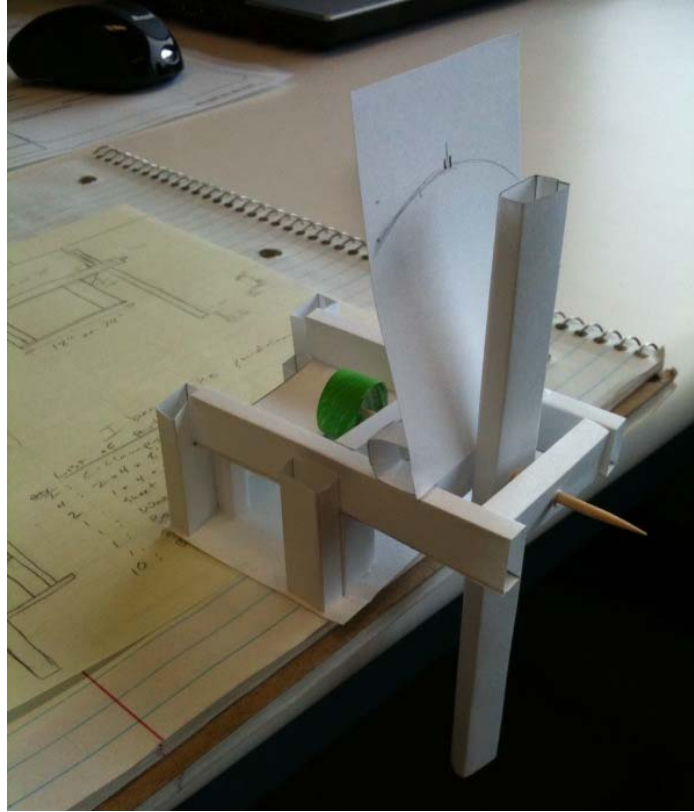


Figure 15. Paper Model of Test Apparatus.

In order to attach the wooden shaft to the encoder, the diameter of the rotating shaft needed to be coupled down from 1 inch to 1/4 inch in diameter. This was accomplished using a 3/4 inch PVC pipe cap with a 3/8 inch hole drilled in the center. A 3/8 inch bolt had the threads cut off with a hacksaw and was fitted through the hole and fixed in place using Plastic Welder Epoxy. The 3/8 inch bolt was connected via a k-coupling to another 3/8 inch mount bearing, which in turn was connected to a 1/4 inch shaft and 1/4 inch k-coupling that connected to the encoder. The purpose of the k-couplings was to negate any alignment errors, since the couplings allowed some flexion but kept the shaft turning at the same rate. The cap with 3/8 inch bolt is shown in Figure 16, and the entire shaft assembly is shown in Figure 17.



Figure 16. 1 inch to 3/8 inch Shaft Coupling.



Figure 17. Completed Shaft to Encoder Coupling.

When assembling the test apparatus, the 1 5/8 inch screws were used wherever the 2 x 4s were attached to the Masonite base or top. Whenever two 2 x 4s were being attached to one another, a 3-inch screw was used. Wood glue was used to provide added stability and rigidity on some of the base joints.

A board was mounted directly next to the pendulum, and the path of the pendulum was traced out on the face of the board. To enable repeatable tests Low, Medium, and High marks were made at the same angle on both sides of the pendulum's direction of travel.

A 6-inch by 7-inch board was cut from the Masonite and was drilled so that both sensors could be attached at the same time, even though only one would be connected to

the data collection equipment at a time. This was done so that each sensor would experience the same motion in repeated tests since the mass on the pendulum's end was unchanging. The sensor board was screwed in to the bottom of the pendulum and could be fixed in two different orientations, depending upon the desired data being collected. A drop of wood glue was used each time the board was fixed to the base of the pendulum to provide added strength. A close-up of this board, with both sensors attached, is shown in Figure 18.

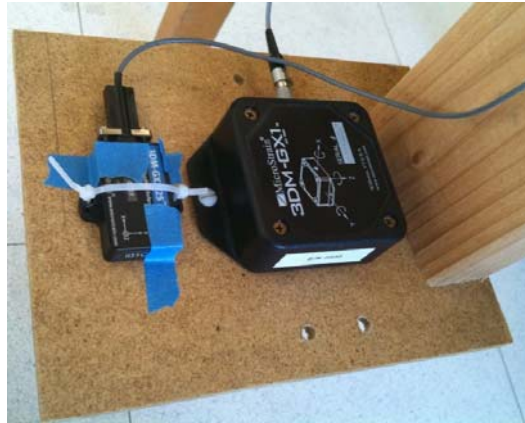


Figure 18. Sensors on Mounting Board, Attached to Pendulum.

Originally, the two MicroStrain sensors were fixed to the board using small metal screws and nuts, but concern arose over potential interference with the magnetic field by having metal so close to the sensors. Two sets of trial data were collected, one using only metal fasteners on the board and the other using only plastic fasteners. The tests were inconclusive, with no clear interference caused by the metal fasteners, but the plastic fasteners performed well enough that they were used rather than risking skewing the data. Plastic screws and nuts from the lab were used to mount the 3DM-GX1 to the board. The 3DM-GX3 enclosure's mounting holes were too small for any plastic screws available in the lab, so zip-ties were used along with simple masking tape to hold it in place.

The final test apparatus is shown in Figure 19, set up in the “horizontal” test orientation, and the final test apparatus is shown in the “vertical” test orientation in Figure 20. Both figures show the “free-swinging” test setup described in Chapter VI.



Figure 19. Final Test Apparatus, Horizontal Test Orientation.



Figure 20. Final Test Apparatus, Vertical Test Orientation.

1. Apparatus Design With Impact Attachment

In order to run the tests that included an impact, the test apparatus had to be modified. A cushioned impact arm was fixed to the test apparatus, as shown in Figure 21 with the additions shown in red. An old running shoe was used to provide the cushion on the 2 x 4 impact arm, and the arm was fixed to the test apparatus in two places using woodscrews. The picture of the test apparatus configured to collect the tests with impact is shown in Figure 22.

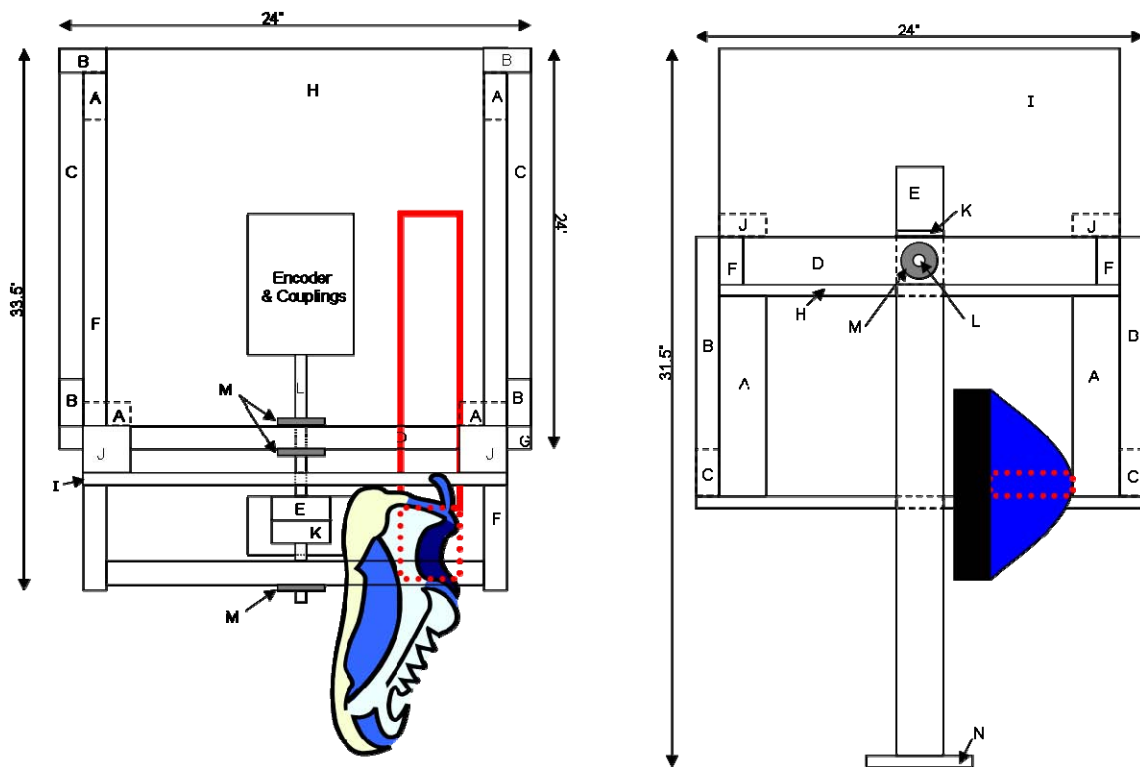


Figure 21. Top and Front View Drawings of Test Apparatus With Impact Board.



Figure 22. Test Apparatus Configured With Impact Board.

2. New Apparatus Cost

The approximate final costs of the materials used in the new test apparatus are shown in Table 4. Numbers have been rounded to the nearest \$0.50 increment if the item's unit cost was more than \$1, and certain materials like the glue and woodscrews screws were included even though the test apparatus did not require the full quantity that was purchased. A miscellaneous hardware amount was included to cover small hardware items used from the lab such as zip-ties and mounting screws. Overall, the cost breakdown shows that the goal of making an affordable new test apparatus costing around \$100.00 was met.

Table 4. New Test Apparatus Cost Breakdown.

Item	Qty	Approximate Cost (ea)	Total
2 in x 4in x 8 ft	4	\$2.50	\$10.00
4ft x 8ft Masonite Board	1	\$10.00	\$10.00
1in Wooden Dowell	1	\$3.50	\$3.50
3/8 in K-Coupling	1	\$25.00	\$25.00
1/4 in K-Coupling	1	\$20.00	\$20.00
Flange Mount Bearing	3	\$10.50	\$31.50
PVC Pipe Cap	1	\$0.80	\$0.80
3/8 in Bolt	1	\$0.80	\$0.80
Epoxy	1	\$5.00	\$5.00
Box of Woodscrews	1	\$6.00	\$6.00
Wood Glue	1	\$3.00	\$3.00
Misc. Hardware	1	\$5.00	\$5.00
		Total Cost:	\$120.60

G. SUMMARY

In this chapter, the rationale and design considerations for a new test apparatus was presented. The detailed specifications for the apparatus were presented and the completed test apparatus was shown. Finally, the approximate costs of the materials making up the test apparatus were provided.

In the next chapter, how the data collection challenges were met is discussed. The hardware used to interface with all of the sensors and data collection equipment is introduced and the software that was used to collect and store the data is presented. Step by step instructions are provided for how to collect and save data. Finally, a graphical user interface is shown that enabled simple data processing once a test had been performed.

V. DATA COLLECTION

The methods used to acquire data for this project are presented in this chapter. Specific hardware and software are discussed and details are provided that show exactly how the different interfaces were set up. The reasons behind the choice of the data collection system architecture are also provided. Truth data generation from the encoder is discussed, and issues related to data collection, such as timing, are presented in detail. Finally, the post-test data analysis software using a MATLAB graphical user interface (GUI) is shown.

A. ENCODER

The Gurley Precision Instruments 16-bit absolute encoder, model A58, was used for this thesis. The specific part number was A58S16MGTT05SAT39Q04. It is the same encoder that was used in [6]. Since it used 16-bits to encode its angular position, the resolution was

$$\left(\frac{1 \text{ full rotation}}{2^{16} \text{ counts}} \right) \left(\frac{360 \text{ deg}}{1 \text{ full rotation}} \right) = 0.00549 \text{ deg/count}. \quad (4)$$

The 16-bit word with the encoder position was broken up into two bytes. The encoder could output one of two data bytes at a time, depending upon what was prompted. When prompted for the least significant byte by setting the output enable one (OE1) pin low, the 8 data pins D0 – D7 represented the position LSB if the data available (DA) pin was reporting high. Similarly, by setting the output enable 2 (OE2) pin low, the encoder provided the most significant byte on D0 – D7 when it was reporting data available. In Table 5, the pin-out of the A58 encoder is shown, while in Figure 23 a graphical representation of the timing diagram for one sample of data requested is shown.

Table 5. Encoder Pin-Out. From [24]

Electrical Signal	Pin	Color
D0	1	yellow
D1	2	brown
D2	3	green
D3	4	yellow-white
D4	5	blue
D5	6	white
D6	7	violet
D7	8	gray
DA	9	white-green (data availability)
OE1	10	red-blue (output enable 1)
OE2	11	pink (output enable 2)
	12	yellow-brown
0 V	13	black
+V	14	red
CASE	15	shield

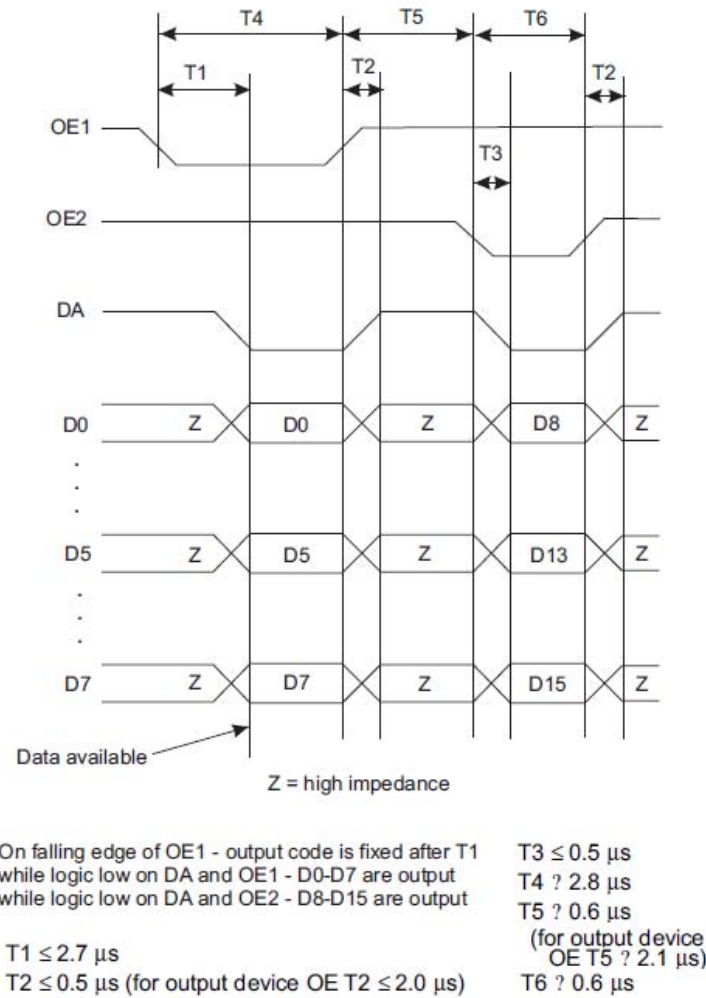


Figure 23. A58 Encoder Timing Diagram. After [24]

A special connector was made in order to connect the encoder to the LabVIEW data collection equipment. The connector connected pins D0 – D7, DA, OE1, and OE2 to corresponding pins on an old parallel port cable that was connected to the data collection equipment. The connector also connected a +5V DC power supply and ground to pins 14 and 13, respectively. Pin 12 was not defined in the documentation and was left unconnected.

B. LABVIEW

National Instruments' LabVIEW 2009 was chosen to be the main data gathering software used for this thesis. LabVIEW was chosen because it could be used to easily create virtual instruments (VIs) that interface between real-world data gathering systems and a computer interface for data processing. LabVIEW used a block diagram programming method as opposed to the more familiar text based programming languages, which allowed for rapid development of data collection programs. Additionally, some VIs were already available to serve as templates for communicating with the encoder and the MicroStrain sensors.

LabVIEW was used primarily for data gathering and storage and, therefore, very little effort was expended making the "front panels" show anything more than indications of proper functionality. The bulk of the programming effort went into creating streamlined and efficient block diagrams that collected and stored data. For data manipulation and presentation MATLAB R2008b was used, as described in detail in a later section.

Initial tests using LabVIEW were run with the 3DM-GX1 and encoder on the old apparatus from [6] using existing VIs from previous unpublished work by James Calusdian. A laptop computer was used to run the VIs and collect the data, the 3DM-GX1 was connected to the laptop via a USB to serial port converter, and the encoder was connected to a NI USB 6501 digital input/output module via USB.

Certain limitations were noted when using just LabVIEW from a laptop computer. First, due to the way the VIs were implemented and because of the laptop computer's limitations, the frequency of the data collected varied somewhat. On average, one sample from the encoder and one sample from the 3DM-GX1 were collected every 35 ms. An attempt was made to increase the sample time but did not yield significant improvements. Second, there was ambiguity when attempting to synchronize the timing. The exact time that the encoder data was pulled was not able to be accurately time-stamped, and this made attempting to synchronize the encoder time with the 3DM-GX1 time somewhat arbitrary. In addition, it was desired to increase the sampling rate of the encoder to at

least 1 kHz, and this was not possible when using LabVIEW on the laptop. For these reasons, and since timing errors on the order of a few milliseconds were considered significant, it was determined that a dedicated system was needed for real-time data collection.

C. COMPACT RIO WITH FPGA

The National Instruments CompactRIO Real-Time controller, the NI cRIO-9012, with a field programmable gate array (FPGA) running at 40 MHz was used to increase the speed of the encoder data collection and to synchronize up all data collection efforts. The CompactRIO had a standalone processor that ran its own operating system, which was optimized to run LabVIEW virtual instruments with far less overhead than a personal computer (PC) operating system. This enabled more predictable timing as well as synchronizing the data collection effort.

The CompactRIO interfaced with a desktop PC via an Ethernet connection through a gateway. This allowed the VIs to be developed on the desktop PC and then deployed to the CompactRIO for the time-critical execution. LabVIEW's Real-Time Environment enabled variables to be shared across the Ethernet connection, and all "button pushing" required to collect data could be accomplished via a single PC.

For applications where timing was extremely critical, like reading the encoder, the CompactRIO had a Field Programmable Gate Array (FPGA) that could be specially configured. This required the LabVIEW Real-Time project module be installed. To configure the FPGA a VI simply had to be created with the FPGA as the target. When building the VI in LabVIEW, this limited the functions that could be used to those which could specifically be implemented using the FPGA. Once the VI was built, LabVIEW automatically compiled the VI and configured the FPGA prior to running. Compiling and configuring the FPGA took up to an hour or more for the VI used in this thesis. However, the long compilation/configuration process was only required one time if the VI was not modified. On subsequent times when the VI ran on the FPGA, it worked like the other VIs on the CompactRIO. For this thesis, the FPGA was used to control and collect the encoder data.

The real-time project structure used was based on one of the basic templates LabVIEW provided. From a top-level view, the CompactRIO was used to collect and buffer the data, and then periodically the data were passed to the host PC to be saved to disk. More specifically, a VI was built for the FPGA on the CompactRIO to gather the encoder data. These data were accessed by a “target” VI running on the CompactRIO. The “Host” VI running on the PC collected the buffered data and wrote to a file on the PC. Global variables were used to make the data accessible between the VIs. The project structure is shown in Figure 24.

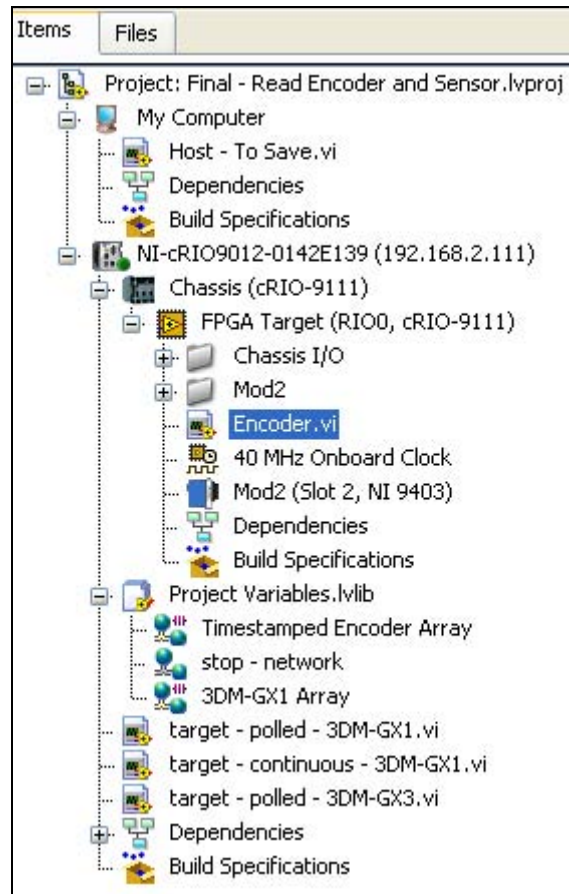


Figure 24. LabVIEW Real-Time Project Structure.

The CompactRIO provided a number of ways to interface with real-world data systems. The Ethernet port was used to communicate with the PC that ran the VIs and collected the data for later use. One serial port provided standard RS-232 data communication. By design, the USB port was configured only for data storage, not for

communication with other data collection units that used a USB. For this thesis, the USB storage was not used due to the Department of Defense Information Systems' ban on flash media and because the Ethernet connection provided an adequate solution for data storage. Finally, four bays were available for compatible National Instruments input/output modules. These I/O modules communicated with the CompactRIO and were able to be accessed directly using the FPGA.

The CompactRIO is shown in Figure 25 with the power and Ethernet cables connected and the COM port and USB connectors empty. In addition, the data collection device used to communicate with the encoder is also seen in the second of the four bays. The CompactRIO was configured with a National Instruments NI 9403 data collection device that was used to communicate exclusively with the encoder. The 3DM-GX1 and 3DM-GX3 used the single serial port to communicate with the LabVIEW VIs.



Figure 25. CompactRIO With NI 9403.

1. NI 9403 Data Collection Device

The National Instruments NI 9403 32-Channel TTL Digital Input/Output Module provided 32 digital input and output ports, which was more than enough for

this project. The FPGA on the CompactRIO was able to control the NI 9403 digital I/O ports directly, enabling very high read and write times.

The pin-out diagram that was used for connecting to the encoder is shown in Figure 26. Note that these were connected via the modified parallel cable discussed in a previous section.

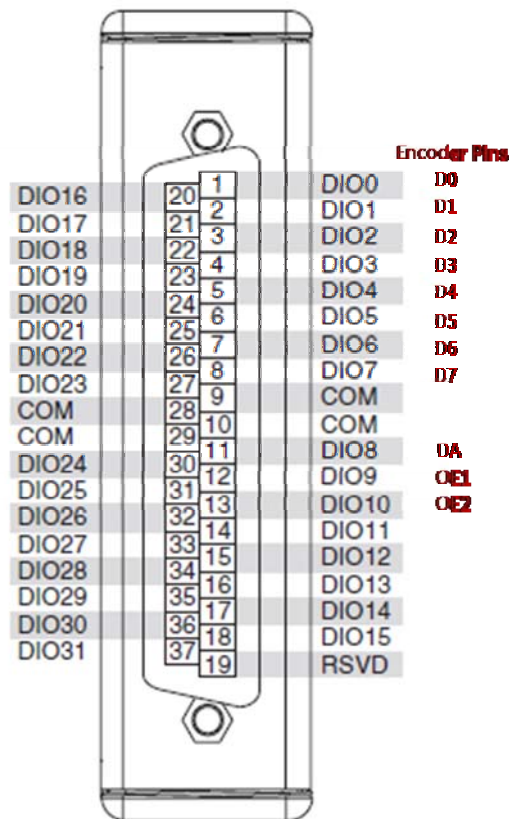


Figure 26. NI 9403 Connector Pin-Out to Encoder. After [25]

2. Encoder.VI

The purpose of the “Encoder” virtual instrument was to collect the encoder data at a fairly high data rate (≥ 1 kHz). The order and timing of the data read protocol shown in Figure 23 was implemented using a VI on the FPGA. Input and output pins were made available via the NI 9403 and were configured to allow the FPGA to have direct access to those lines. This is shown in Figure 27. The items D0 through OE2 in the Mod2 folder beneath the FPGA target were all directly accessible by the FPGA.

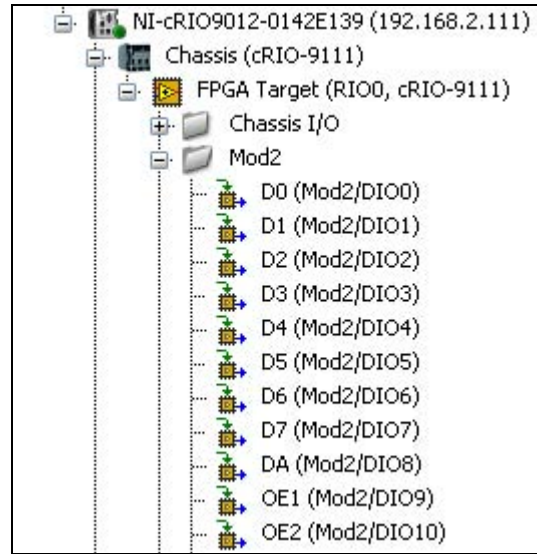


Figure 27. FPGA–Direct Access to Encoder Data Pins.

From a top-level perspective, the encoder VI operated on a fixed cycle to read one sample of encoder data per cycle, time-stamp it, and then buffer that data. Each cycle the LSB and MSB data from the encoder, along with the clock “tick” from the FPGA clock when the LSB was pulled, were buffered. In Figure 28, the block diagram of “Encoder.vi” is shown.

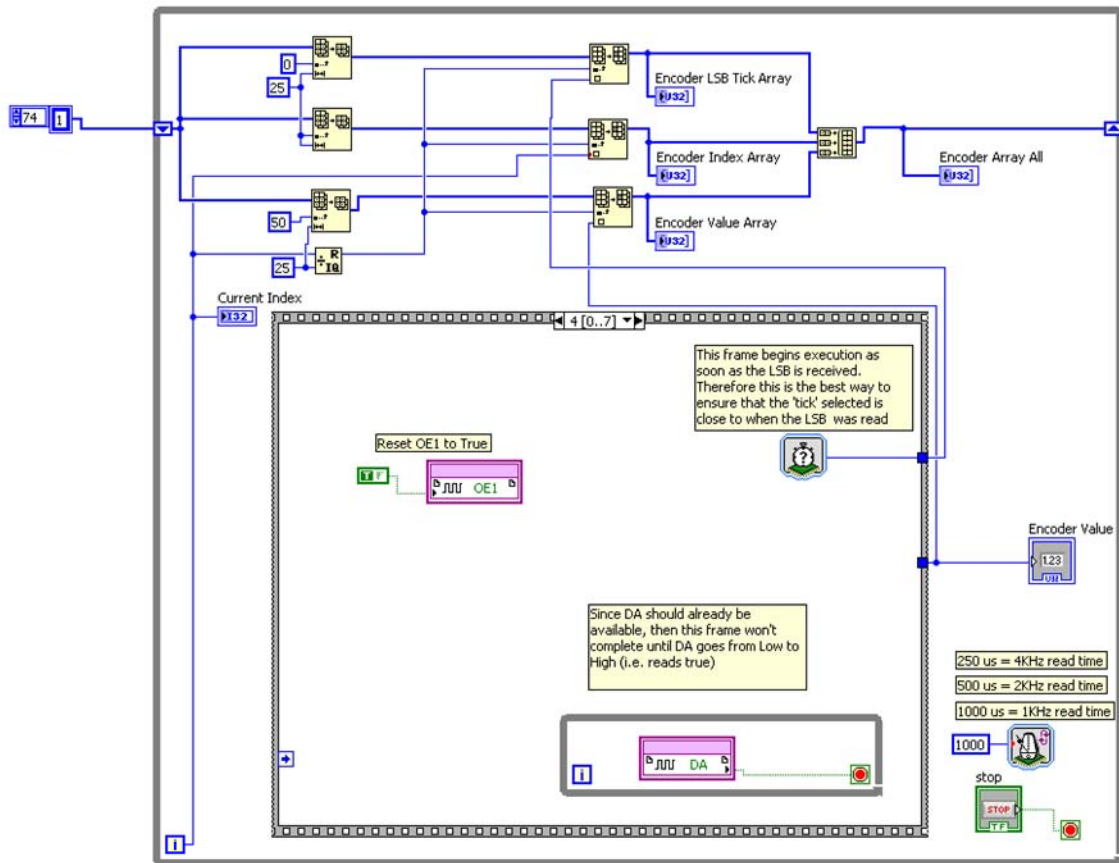


Figure 28. Encoder.VI Block Diagram.

A one-dimensional 75-element array was used to buffer the data. The idea was that the encoder VI would be running at a much higher rate than it was called to output data, so the data values collected each cycle were buffered to ensure no data were lost. This created the potential of pulling the same data multiple times, depending on how quickly the data were pulled. At 1 kHz, if this array was read faster than once every 25 ms, then some data would overlap. Conversely, if it was read slower than once every 25 ms, then some data would be lost. Care was taken to ensure that, as much as possible, no data were lost, since removing redundant data was a simple thing to accomplish in post-processing.

The 75-element buffer array was broken into three sub-arrays. The first sub-array, indexes 0–24 of the overall buffer, stored the previous 25 values of the tick time that the encoder least significant byte was received. The next sub-array, indexes 25–49 of the

overall buffer, was for the overall index of the encoder VI loop. This value counted up from zero as soon as the VI started and continued upward as long as the VI was running. The index value was important since the tick time could rollover while the loop index would not. Finally, indexes 50–74 of the overall buffer stored the previous 25 values of the encoder position, an integer from 0 to 2^{16} , referred to as the encoder “count.” In each sub-array, the index that was overwritten incremented by one every cycle, modulo 25, such that the index in each sub-array depended on the current iteration of the loop (see Figure 28).

A stacked sequence was used to implement the required sequence of events to collect data from the encoder. The sequence has been expanded out and is shown in Figure 29. Note that the magenta boxes each represented a single pin on the NI 9403, as labeled in Figure 27. The data were collected LSB first, then MSB, in accordance with [24]. Since the LSB was the most critical values and the MSB was unlikely to change significantly over a small period of time, the time that the LSB read was completed was the tick time collected and associated with that particular value.

The maximum data rate attempted with the block diagram shown in Figure 28 was 4 kHz. This was demonstrated successfully but ultimately 1 kHz was used because of timing concerns. At 4 kHz the buffered data array would have to be read and stored once every 6 ms, and this was not realizable with the overall design. An adequate number of samples from the encoder and data were able to be collected at 1 kHz without significant loss of samples.

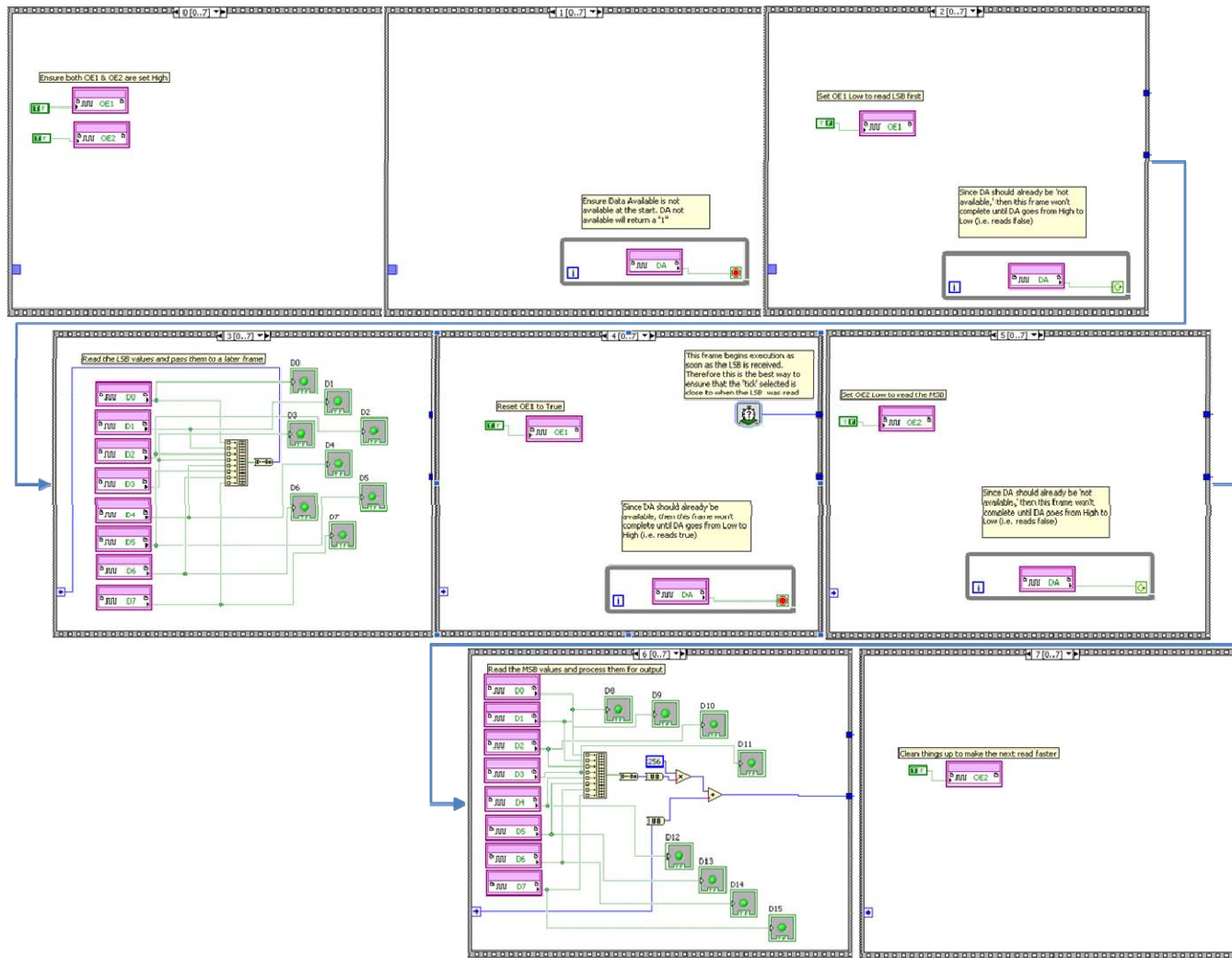


Figure 29. Encoder.VI Stacked Sequence, Broken Out.

3. Target.VI

The “target” VIs were deployed to the CompactRIO, and their goal was to interface with the encoder VI as well as with the MicroStrain sensor and then make that data available over the Ethernet connection. A separate VI was written for communication with the 3DM-GX1 and the 3DM-GX3. Both target VIs were built to communicate with the MicroStrain sensors in a “polled” manner, as described in Chapter II. The block diagram was very similar for each and, therefore, only the 3DM-GX3 target VI is shown in detail in this section while the full block diagram for the 3DM-GX1 and the corresponding details are found in Appendix A. When applicable, major differences between the two VIs will be highlighted and expanded upon in this section.

The VI “target - polled - 3DM-GX3.vi” was built to run two timed loops in parallel, one to call the “encoder.vi” and collect data from the encoder, and the other to collect data from the 3DM-GX3 via the serial port. In this section Loop 1 refers to the timed loop in which the encoder data were collected, while Loop 2 refers to the timed loop in which the 3DM-GX3 data were collected.

Both loops buffered the data received and periodically updated a global variable with the new data. The timing of each loop was adjusted empirically to ensure adequate timing for data collection. If the loops were executed too fast, certain threads would out-prioritize others on the CompactRIO, which resulted in gaps in the data. Similarly, attempting to update the global variables each loop iteration caused the entire VI to suffer a performance decrease resulting in gaps in the data. For best performance Loop 1 and Loop 2 were iterated once every 10 ms.

a. Loop 1

For Loop 1 the encoder.vi was set to run continuously on the FPGA. Once per cycle the encoder data would be pulled and buffered. A two-dimensional array buffered 100 samples of encoder data. Each sample of encoder data contained 75 data points corresponding to the 25 previous encoder values. These values were from the buffer array generated by encoder.vi. In addition to those 75 points, the time-stamp and

encoder loop index corresponding to when the encoder data were read were also buffered with the data each loop iteration. Every 100 cycles the buffered array was converted to a one dimensional array and then output to the global variable “Timestamped Encoder Array.” The global variables were required to be one-dimensional due to LabVIEW convention.

Every time the encoder data were pulled the onboard CPU time was called and recorded. This provided a reference time for the value of the encoder index at the time that the encoder data array was pulled, within a few tens of microseconds. Thus, a reasonably accurate method of time-stamping the encoder data to a common reference time, the CompactRIO clock, was established. The overall process is shown in Figure 30.

In Figure 30, it is seen that there was a conditional case structure that handled updating the global variable. The “True” frame collected the encoder data and time-stamp, updated the buffer array, and then updated the global variable. This frame executed once every 100 iterations, which worked out to once per second with a 10 ms timed loop. The “False” frame simply collected the encoder data and time-stamp and then updated the buffer array. This process is shown in Figure 31.

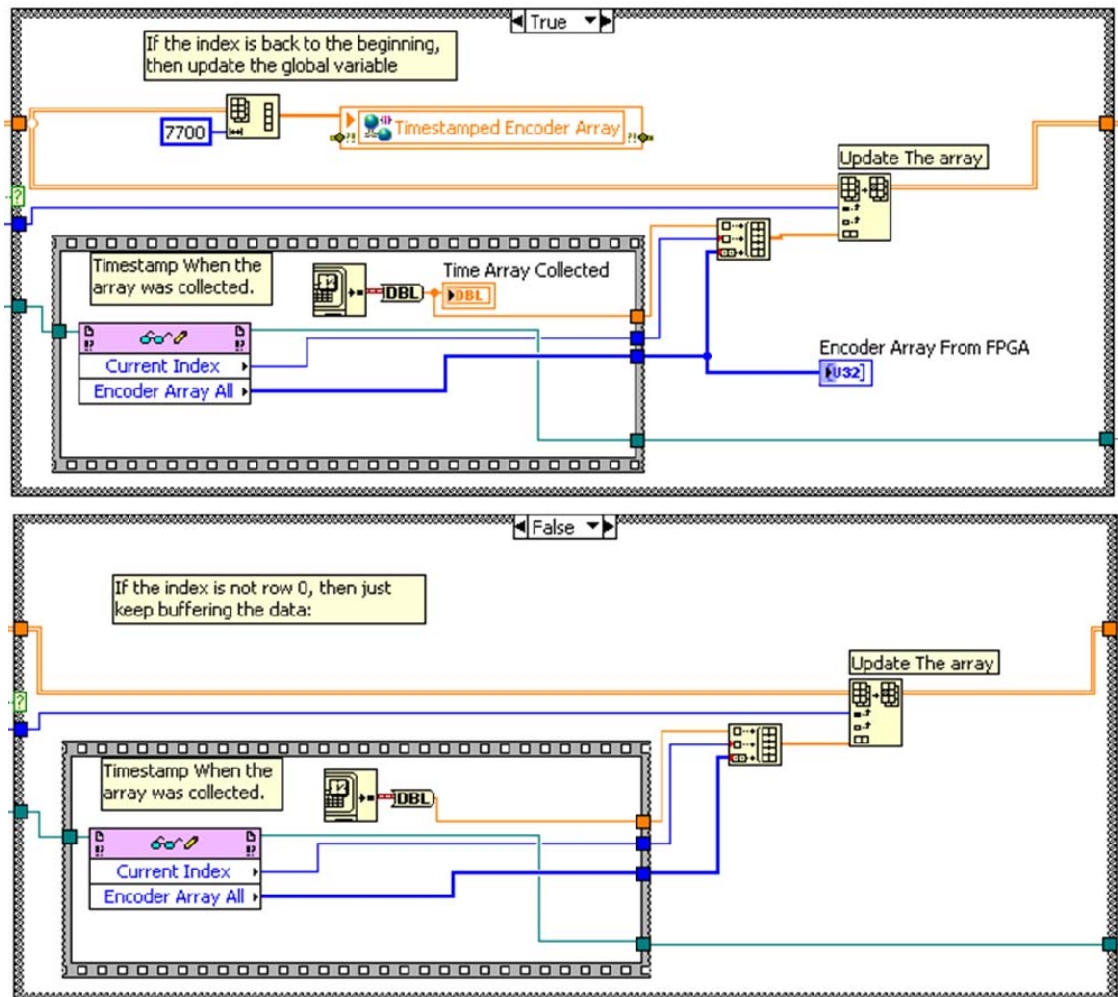


Figure 31. 3DM-GX3 Target VI, Loop 1, True/False Frames Expanded.

b. Loop 2

Loop 2 collected the data from the MicroStrain sensor, buffered it, and then periodically updated a global variable with the buffered sensor data. To accomplish this, first the serial port was initialized for communication with the 3DM-GX3. The 3DM-GX3 defaulted to a baud rate of 115,200, whereas the 3DM-GX1 communicated with a baud rate of 38,400. With each cycle, the encoder was commanded to output a certain set of data based on [14] or [16], and then the data were collected. The exact details of this operation differed between the 3DM-GX1 and the 3DM-GX3 but, in effect, the output of both was an array with the data blocks available for further processing. The overall LabVIEW block diagram for loop 2 is shown in Figure 32.

The specific sub-VIs that were used to command, pull, and convert the GX1 and GX3 data were: “Send 3DM-G Data Request Command.vi,” “Get Record From Serial Port.vi,” and “Convert Serial Record to Integer Array.vi,” for the 3DM-GX1 and “Send 3DM-GX3 Data Request Command.vi,” “Get 3DM-GX3 Record From Serial Port.vi,” and “Convert 3DM-GX3 Record to Integer Array.vi,” for the 3DM-GX3. The block diagrams for these VIs are shown in detail in Appendix A.

A time-stamp was collected each time the data were read in from the MicroStrain sensor. There was an inherent latency between the time the data request was received, processed, and then output to the serial port by the MicroStrain sensor and the time they were received by the VI on the CompactRIO. This time misalignment was handled in the data post-processing, since there was no way to synchronize the GX1 clock to the CompactRIO clock. There appeared to be a way in [16] to synchronize the GX3 clock to the CompactRIO clock using the “Timer” command, but that was not investigated for this thesis due to time constraints.

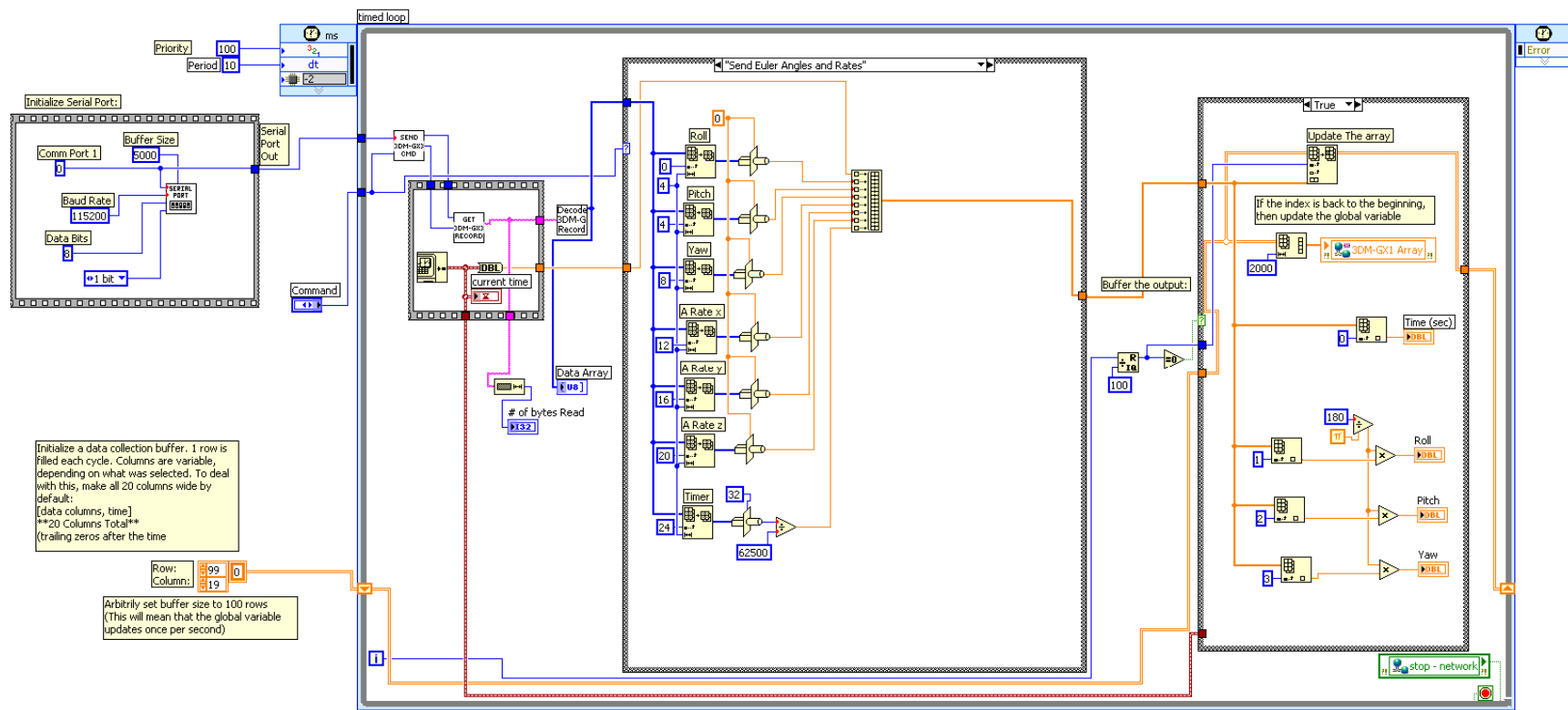


Figure 32. 3DM-GX3 Target VI, Loop 2.

The exact format of the data differed between the 3DM-GX1 and the 3DM-GX3. The 3DM-GX1 data returned were nothing more than integers that required scaling according to [14] to convert the values into meaningful results. The data returned from the 3DM-GX3 were usually 32-bit single precision floating point numbers conforming to IEEE-754. However, some data that could be requested from the 3DM-GX3 were not 32 bits long, and for flexibility the data array after processing the serial data was simply an array of bytes. These bytes were then combined as necessary in the VI to produce meaningful results. A sample of this recombination process is shown in Figure 33, where the Euler angles and Euler angle rates are recombined and converted to single precision floating point numbers in accordance with [16]. The Euler angles were in the Earth/Navigation reference frame. Also note that a timer value was returned by the GX3, representing the number of seconds after power on that the data was valid. Most often the data requested for the GX3 were the Euler Angles and the Euler Angle Rates. However, a few other data requests were built in and are shown in detail in Appendix A.

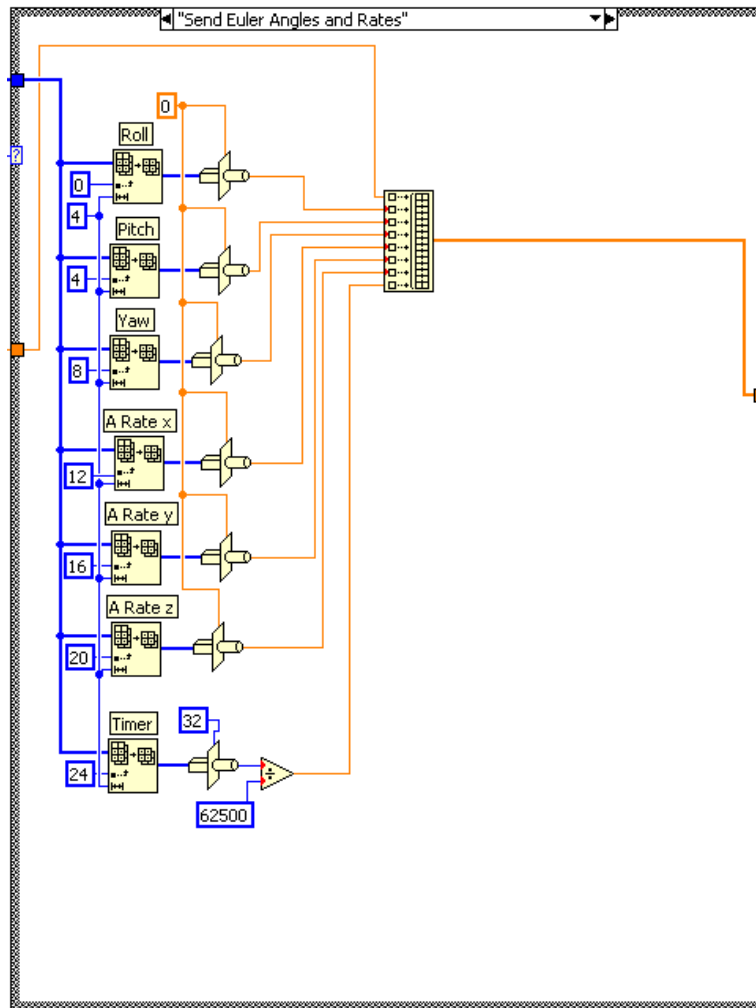


Figure 33. 3DM-GX3 Target VI, Data Recombined for Euler Angles and Rates.

For the 3DM-GX1, the only data pull built into the VI was for the Gyro-Stabilized Quaternion and Vectors, which returned the quaternion, x , y , z magnetic field, x , y , z acceleration, and x , y , z angular rate. All x , y , z were in the body reference frame and converted to the Earth/Navigation frame via the quaternion. It also returned a timer value that could be converted and used in a similar way that the 3DM-GX3 time was. The frame that converted the serial data integer array from the 3DM-GX1 into the desired values is shown in Figure 34.

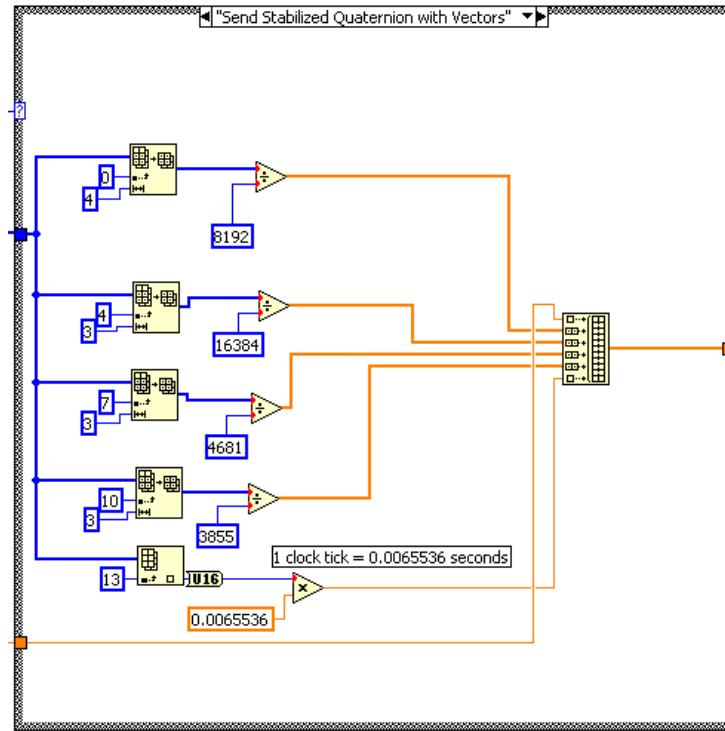


Figure 34. 3DM-GX1 Target VI, Recombined Stabilized Quaternion and Vectors.

The scaled or combined data were then passed to a buffer, along with the time-stamp. A two-dimensional array of 20 rows and 100 columns was used to buffer the data. The first row contained the time-stamp, and the next rows contained up to 19 different outputs. If a certain data request did not have 19 outputs, then the remaining rows were simply 0 valued. The index of the buffer was incremented every iteration of the loop and was tied directly to the loop index, modulo 100.

The conditional structure that controlled the buffering and updating of the MicroStrain sensor data is shown in detail in Figure 35. When it was time to update the global variable “3DM-GX1 Array” the two-dimensional buffer array was flattened to a one-dimensional array because of the requirements of LabVIEW. Note that in order to simplify the data collection and have as few VIs as possible to collect and save the data, the global variable “3DM-GX1 Array” is used to represent both the 3DM-GX1 and the 3DM-GX3 data since only one of the sensors can be connected to the CompactRIO at a time.

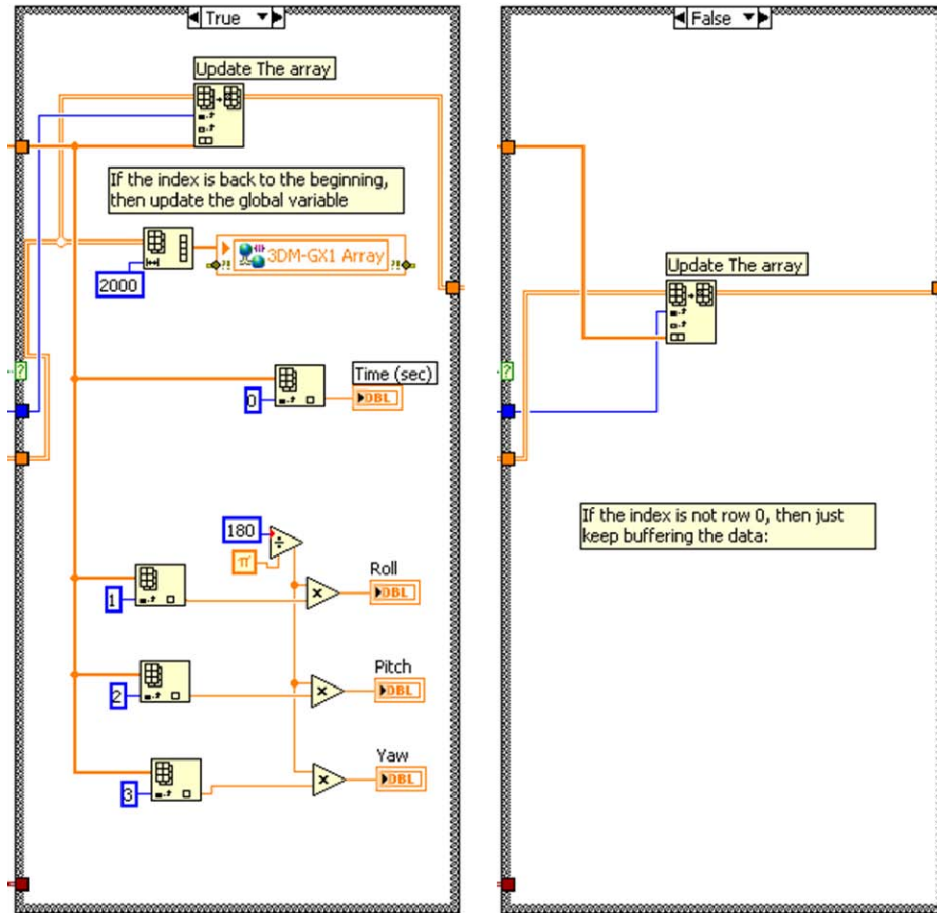


Figure 35. 3DM-GX3 Target VI, Loop 2, True/False Frames Expanded.

4. Host-to-Save.vi

The “Host-to-Save.vi” ran on the host PC and was used to read the data collected by the CompactRIO and save it to disk. The timed loop executed once every 10 ms and continually read in the two global data variables “Timestamped Encoder Array” and “3DM-GX1 Array,” writing them to file when either variable was updated. The block diagram for this VI is shown in Figure 36.

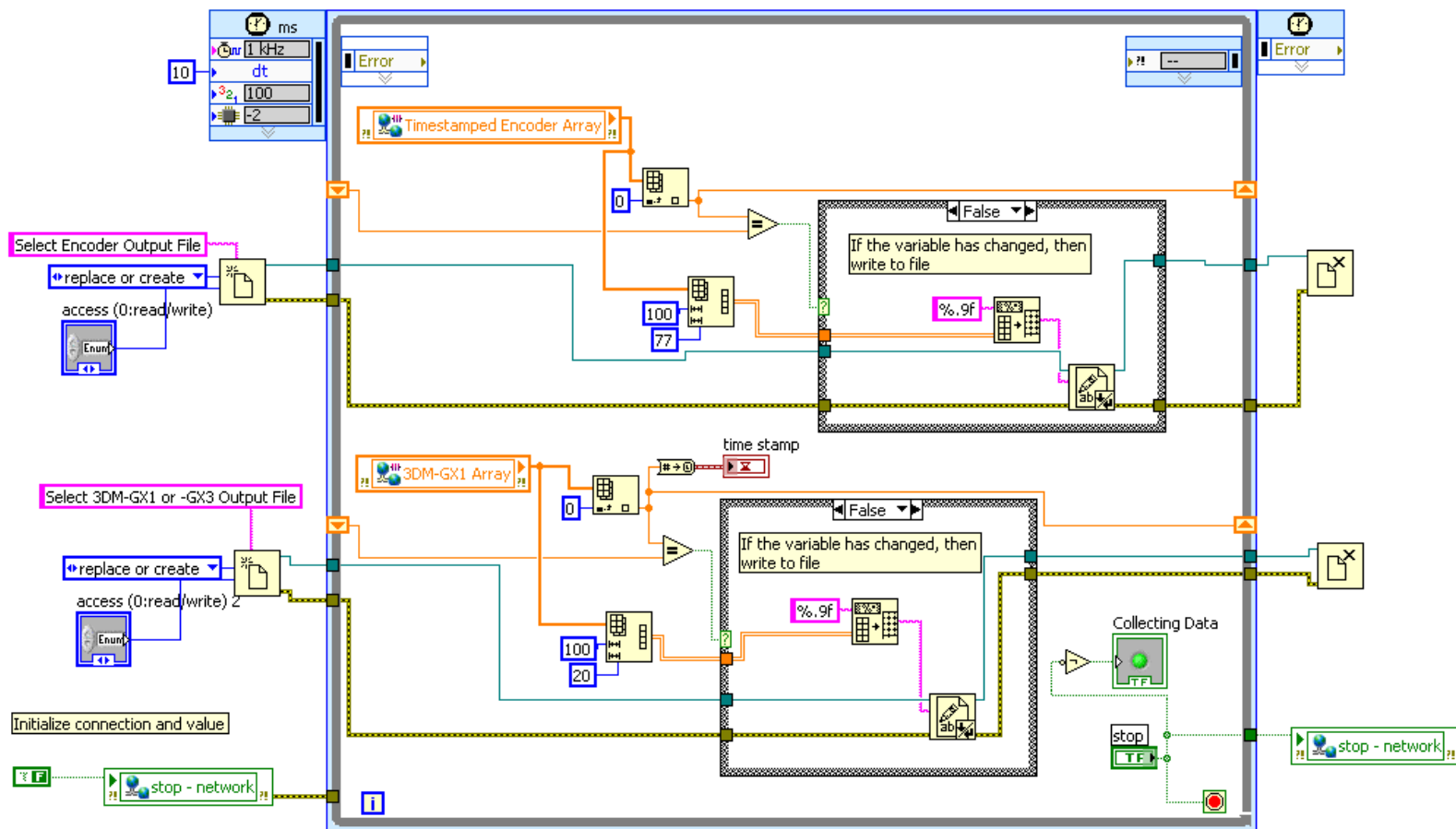


Figure 36. Host-to-Save.vi.

Upon execution, the VI prompted the user to input two file names, one for the MicroStrain sensor data and one for the encoder data. Then, the global variables were read in and converted back into two-dimensional arrays, a 77 row by 100 column array for the encoder data and a 20 by 100 array for the MicroStrain sensor data. If the data arrays had been updated, they were then appended to their respective tab delimited text file. If the arrays had not been updated, then no action was taken and nothing was written to file. Stopping the execution of this VI stopped the execution of the other VIs.

5. LabVIEW Data Collection Procedures

The following outlines the proper procedures for collecting MicroStrain sensor data with the corresponding encoder data. The following steps should be completed in order:

- Ensure power is on. Power should be checked on the CompactRIO, the 5 V dc power supply to the Encoder and either the 3DM-GX1 or 3DM-GX3.
- Run the appropriate target VI, either “target - polled - 3DM-GX1.vi” or “target - polled - 3DM-GX3.vi” and make sure everything on the front panel appears to be collecting data. The numbers in the array should be non-zero, the encoder counts should be non-zero, and it is expected that some numbers will change at least every second, even while at rest.
- Run the “host - to - save.vi,” and when prompted enter the name of the text file for the sensor data and the name of the text file for the encoder data. Be sure to include the extension “.txt” in the file name.
- Conduct Experiment. Experiment length should be kept to approximately 30 seconds or less due to data processing limitations.
- Hit stop on the host front panel. If target is still going, hit stop on the target (this was required occasionally). A known bug is that if target stop button was used instead of the host, then the next time the target was run it

would not run continuously and would stop almost immediately. The next time the VI was run it would work as expected.

D. ORIENTATION DATA CONVERSIONS

The data collected from the encoder representing the shaft position were the raw “counts,” which were integers that increased from 0 to 2^{16} as the shaft turned clockwise through one full rotation. These counts were converted to degrees using (4). When referenced to the initial value collected, the encoder data collected represented the angular displacement from the starting point.

The encoder data could represent the roll, pitch, or yaw in the Earth/Navigation reference frame, depending upon the orientation of the sensors and the test apparatus pendulum. With the pendulum vertical and the MicroStrain sensor’s y -axis aligned with the direction of travel, the encoder data represented roll. Note that if the sensor’s x -axis was pointing in toward the encoder, then the encoder angle represented roll directly, but if the sensor’s x -axis was pointed away from the encoder, then the encoder angle represented $(-1)(\text{roll})$ and had to be appropriately scaled in the data analysis. When the pendulum was vertical and the sensor’s x -axis was aligned with the direction of rotation, then the encoder angle represented the pitch. Finally, when the entire test apparatus was tilted up 90 degrees and the pendulum movement was parallel to the ground and the x -axis of the sensor was aligned with the pendulum’s direction of travel, as in the case for pitch, the encoder data represented yaw.

The orientation data collected from the 3DM-GX3 were already provided as Euler angles that were referenced to the Earth/Navigation reference frame and could be used for roll, pitch, and yaw. Although the 3DM-GX1 had the ability to output these angles directly as well, previously developed data collection VIs had focused on providing body frame referenced data and a quaternion,

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}. \quad (5)$$

From this vector the Euler angles representing roll, pitch, and yaw could be calculated [9] by

$$\text{roll} = \phi = \tan^{-1} \left(\frac{2(q_2 q_3 + q_0 q_1)}{q_0^2 - q_1^2 - q_2^2 + q_3^2} \right), \quad (6)$$

$$\text{pitch} = \theta = \sin^{-1} (2(q_0 q_2 - q_1 q_3)) \quad (7)$$

and

$$\text{yaw} = \psi = \tan^{-1} \left(\frac{2(q_1 q_2 + q_0 q_3)}{q_0^2 + q_1^2 - q_2^2 - q_3^2} \right). \quad (8)$$

E. ANGULAR VELOCITY AND ACCELERATION

In addition to measuring the orientation of the sensor it was also desired to test the accuracy of the MicroStrain's angular velocity and linear acceleration outputs, if possible. Both of these parameters were output in the sensor's body reference frame. Each additional parameter required a new "truth" reference for comparison.

1. Theoretical Calculated Truth

To measure the angular velocity, the pendulum must be in the vertical position and the MARG sensor needs to be aligned so that the axis under test is in line and parallel to the axis of rotation and, ideally, horizontal to the ground plane. Aligned this way the encoder data represents the angle turned about the axis under test, and the angular velocity ω , also referred to as the angular rate, can be calculated by

$$\omega(t) = \frac{d}{dt} \theta(t) \quad (9)$$

where $\theta(t)$ is the angle of the encoder over time.

To measure the dynamic accuracy of the acceleration output by the sensors, which was a linear acceleration and not an angular acceleration, basic physics is applied to calculate the truth data using only the encoder angle. For this explanation, assume that the x -axis of the sensor is aligned with the direction of travel of the pendulum, the z -axis of the sensor is in line with the pendulum, and positive z is down with the pendulum positioned vertically. Note that any two axes can be aligned in this fashion for testing.

The total linear acceleration at the end of the pendulum while it is swinging is a combination of the tangential acceleration and the radial acceleration, both due to the pendulum swinging, as well as the acceleration component due to gravity. From [26], the tangential acceleration is

$$a_t = \alpha r \quad (10)$$

where α is the angular acceleration and r is the radius of the pendulum from the center of the axis of rotation to the center point of the sensor. The angular acceleration can be calculated by

$$\alpha(t) = \frac{d^2}{dt^2} \theta(t). \quad (11)$$

The radial acceleration directed toward the point of rotation is

$$a_r = \omega^2 r. \quad (12)$$

Then the total linear acceleration at the end of the pendulum, including gravity, is

$$a_l = a_t + a_r + a_g. \quad (13)$$

Since the body axis was aligned with x in the direction of travel and z aligned with the pendulum, the component accelerations are

$$a_x = a_l + a_g \sin \theta \quad (14)$$

and

$$a_z = a_r + a_g \cos \theta. \quad (15)$$

From (10)–(15), in terms of the encoder angle θ and the radius r only, the linear acceleration of the axis under test is determined by

$$a_x(t) = \left(\frac{d^2}{dt^2} \theta(t) \right) r + a_g \sin(\theta(t)) \quad (16)$$

and

$$a_z(t) = \left(\frac{d}{dt} \theta(t) \right)^2 r + a_g \cos(\theta(t)). \quad (17)$$

2. Actual Calculated Truth

Although the desired signal from the encoder was purely the shaft angle, the actual signal from the encoder had a certain amount of added noise, probably due to

quantization errors in the encoder as well as noise in the system. The presence of noise significantly impacted the ability to calculate accurate “truth” data from the encoder angle.

For example, when released from approximately 70 degrees in a free fall, the pendulum period was approximately 1 second. Assuming this was the highest frequency event that was recorded and that it was allowed to damp out to a stop, then the maximum expected frequency expected would be 1 Hz. Allowing for other influences, we expect frequencies not more than 5 to 10 Hz. Looking at a zoomed in plot of the raw encoder data in Figure 37, we see that the encoder data is not just a smooth low frequency signal since there are several “hops” and “jumps” off of what should be a smooth line recording of the encoder data.

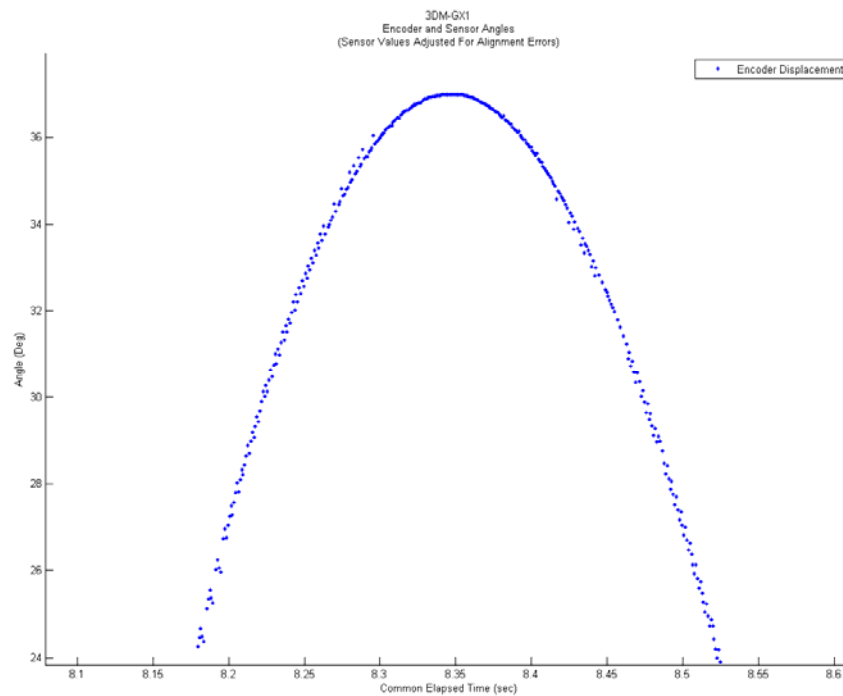


Figure 37. Sample Encoder Data, Not a Smooth Curve.

The fast Fourier transform (FFT) of the encoder data was taken, and it showed that there appeared to be some amount of white noise in the system. In Figure 38, the FFT of the encoder data of the same sample data is shown.

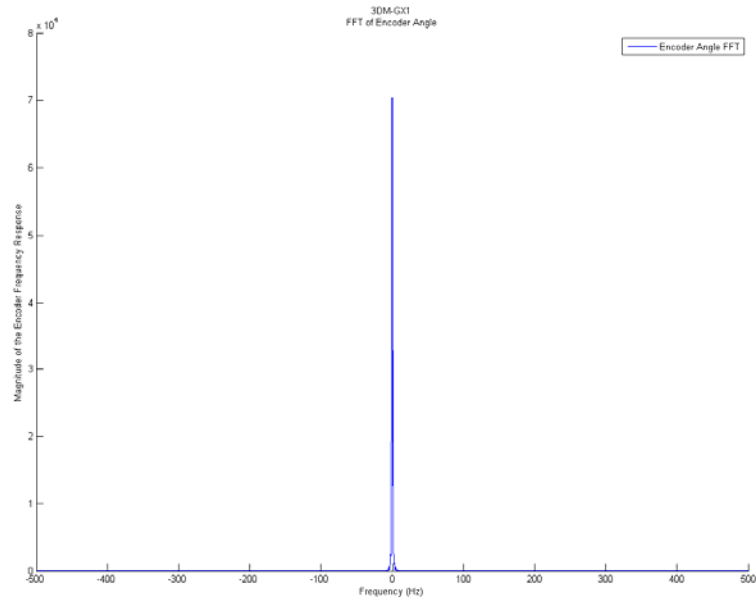


Figure 38. FFT of Sample Encoder Data.

From the figure there appears to be only significant low frequency data. In Figure 39, the same FFT is shown only zoomed in around the frequencies of interest, with peaks near 1 Hz as expected.

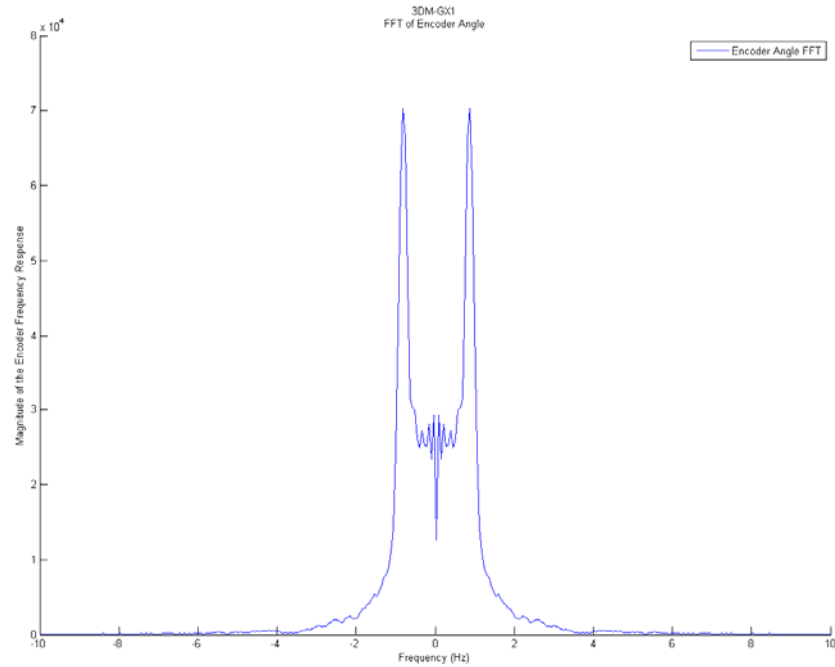


Figure 39. FFT of Sample Encoder Data, – 10 Hz to 10 Hz.

Although the major frequencies in the signal are near 0 Hz, the same FFT is shown in Figure 40 across all sampled frequencies but zoomed in on the noise (the magnitude of the y axis limited from 0 to 25). Note that there are relatively uniform values across all frequencies, indicating some kind of white noise.

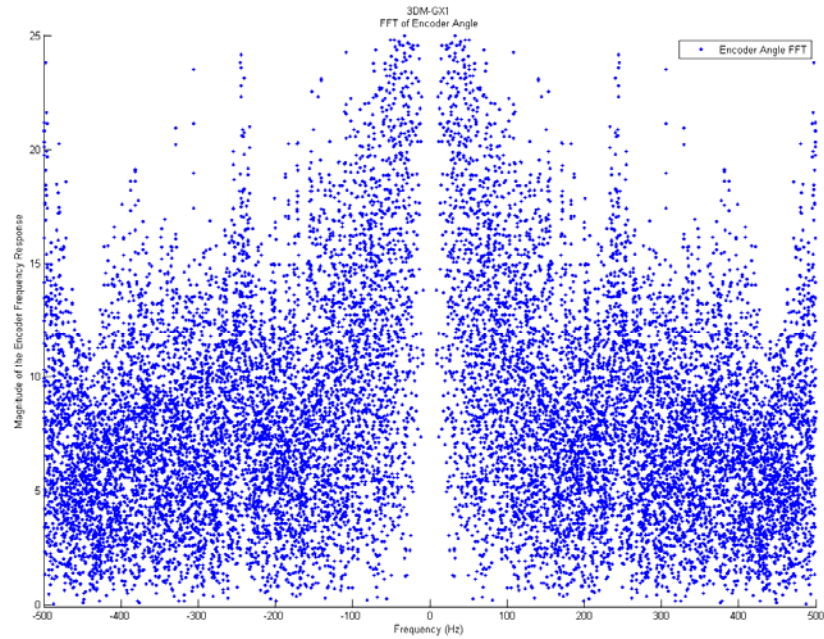


Figure 40. FFT of Sample Encoder Data, ± 500 Hz, Zoomed to Show Noise.

Although not noticeable when computing the accuracy of the angle output by the sensor, the presence of the noise is much more significant when computing the truth data for angular velocity, the time derivative of the encoder angle, from (9). With the same encoder data as in Figure 37, the angular velocity was calculated and is shown in Figure 41, with the sensor's measured angular velocity plotted on top. The two are sometimes aligned, especially at the lower angular velocities, but more often than not the MicroStrain sensor's data shows what the calculated "truth" data should be rather than the other way around.

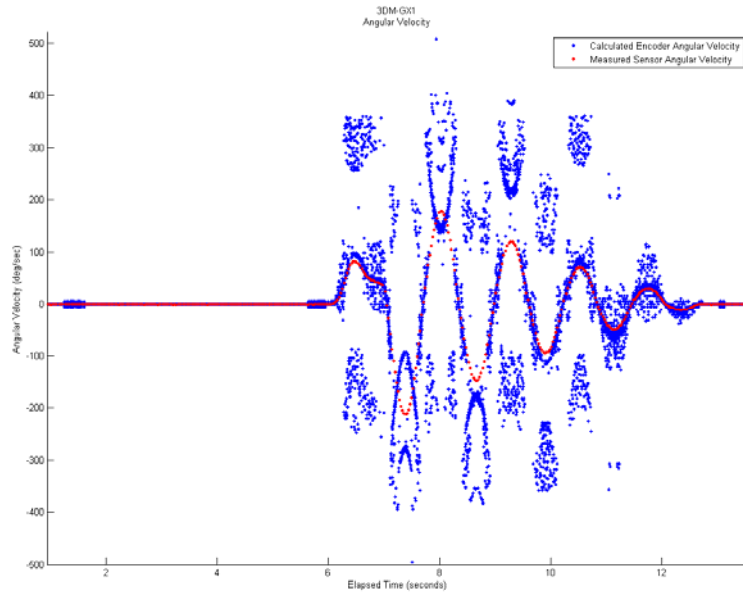


Figure 41. Sample Calculated Angular Velocity.

In Figure 42, the FFT of the derived encoder angular velocity is shown. It is clear that the high frequency noise has been amplified and is no longer insignificant.

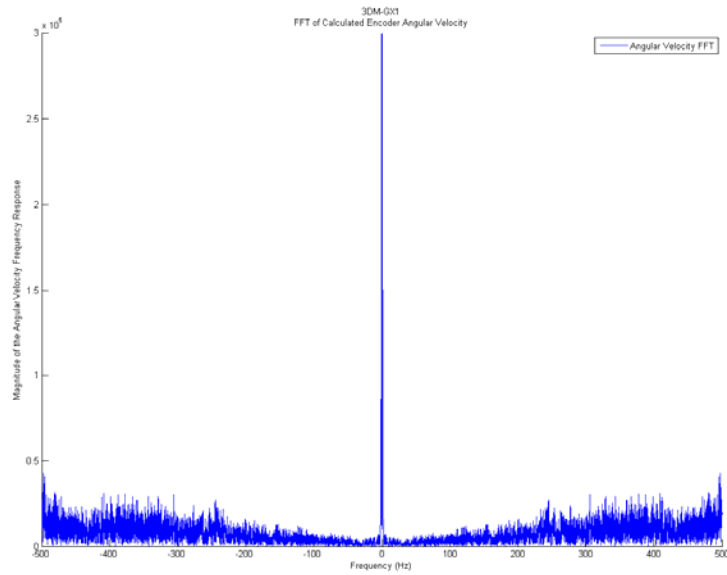


Figure 42. FFT of Sample Calculated Angular Velocity.

In the Laplace domain, differentiation is simply s times the Laplace transform of the signal. Since $s = e^{j\omega}$ it is clear that the higher frequencies are going to be scaled more than the lower frequencies. When the signal is differentiated, the high frequency parts of the noise grow faster than the low frequency parts of the noise. This is most apparent in Figure 43, which shows the FFT of the calculated angular acceleration. This was calculated by differentiating the angular velocity. Although the low frequency “spikes” of the desired signal can be seen, the calculated angular acceleration is dominated by the growing noise, such that the desired low frequency signal in the center is not even recognizable.

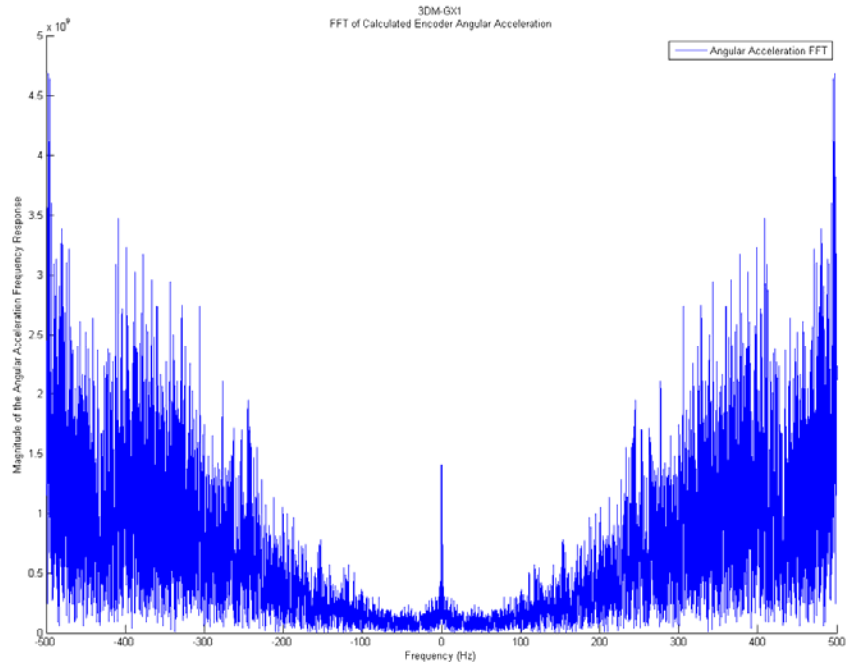


Figure 43. FFT of Sample Calculated Angular Acceleration.

Given Figures 42 and 43, it is clear that the linear accelerations calculated with (16) and (17) are not going to be valid. Thus, without manipulating the data in some way, the angular velocity and linear acceleration “truth” data cannot be calculated from the encoder angle.

3. Calculated “Truth Data” Work-Arounds

Since the problems with the calculated angular velocity and angular acceleration were due to noise, the obvious first solution was to attempt some kind of filter. Early testing was done using a very simple sliding window averaging filter, as well as a more complicated equi-ripple FIR low pass filter with a pass band at 10 Hz, a stop band at 50 Hz, and a 30 dB stopband ripple. Many different methods were attempted. The filter was applied to the encoder data only, to the encoder data and the calculated angular velocity, and, finally, to the encoder data, the calculated angular velocity, and the calculated angular acceleration. The code used for the FIR filter can be found in Appendix B, Encoder_Filter.m. This method met with some success, as seen in Figure 44, which shows the same data from before filtered using the FIR filter previously described on both the encoder data before it was differentiated as well as the calculated data after differentiation.

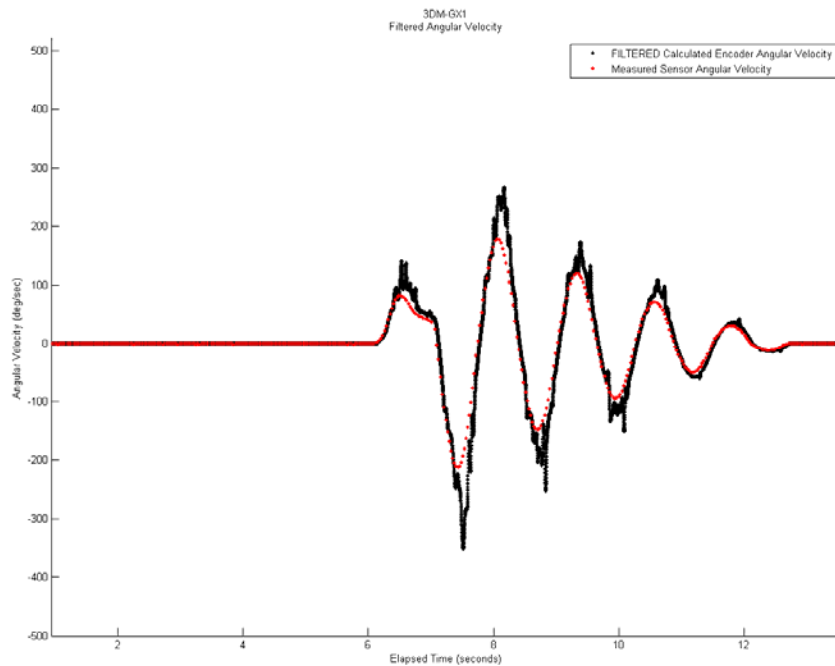


Figure 44. Sample Calculated Filtered Angular Velocity.

Clearly, the filtered data more closely tracks the sensor data than the unfiltered data did, but it is still not accurate enough to use as a truth source for comparison. In addition, the FIR filter scaled the data and slightly shifted it in time. An FIR filter was chosen with linear phase so, theoretically, the timing could be resolved; however, this would require even further manipulation of the “truth” data.

Another possible way to extract the encoder angle from the noise is to implement a Wiener filter on the collected encoder data. In theory, this provides the best estimate of the encoder angle with the least error, which can then be used in combination with differentiation and filtering. This method was not attempted due to time constraints.

Ultimately, all of the solutions required too many manipulations of the data to get the required “truth” data. The more the data were manipulated, the less “true” they became. Without a valid source of truth data, the angular velocity and linear acceleration values were not tested and, thus, for this thesis only the orientation data was tested

F. TIMING

Even though the data were time-stamped or indexed, as mentioned in previous sections, every single point did not have a timestamp in seconds. In addition, there was a small amount of variation in the time-stamping process and, therefore, post processing was required to synchronize the data to the proper times. This synchronization was done using MATLAB during post processing, and the commented code with specific details can be found in Appendix B.

This section discusses generally what was done to get the data referenced to a common clock time and does not expand on the details of the code. The CompactRIO clock time-stamp provided the number of seconds elapsed since 12:00 A.M. Universal Time, January 1, 1904. This time-stamp was used as the common clock reference to which all data were synchronized. Test results were ultimately converted into a common “elapsed time” for simplicity.

1. Data Collection Time-Stamping

The encoder.vi, which ran on the FPGA, did not have direct access to the CompactRIO clock. Instead, it had access to a tick counter that incremented with the FPGA's clock and an index counter that was incremented with each iteration of the VI's execution loop. When the array containing the 25 current and previous values of the encoder was pulled by the Target.VI on the CompactRIO, the current index of the FPGA was also pulled. A time-stamp from the CompactRIO was associated with that particular index, and with knowledge of the FPGA clock ticks and index (1 tick was 1/40,000 of a second, and the encoder index incremented every 0.001 seconds) the tick-stamped encoder data could be resolved to the single time-stamped encoder value. Because the time that the encoder index was read and the time the stamp was pulled can have some small amount of variation (due to LabVIEW constraints), 100 points that had been time-stamped directly were selected at evenly spaced intervals through the data, and then all of the data collected were synchronized with respect to that single time-stamp. The time-stamp of all 100 sets of data were then averaged to provide a uniform time for each encoder value that was referenced to the CompactRIO clock. The details of this process are shown in Appendix B under the function "process_tick_time.m."

For the MicroStrain data, the time-stamp was recorded when the data were received on the serial bus. Since both the 3DM-GX1 and 3DM-GX3 had a timer that indicated when the data were valid, the recorded CompactRIO time-stamp was associated with that time. This introduced a small time offset since it was unrealistic to assume that the time the data arrived at the serial port was the exact time that the data were valid. However, no better solution was available, and the data from the sensors were referenced to the CompactRIO clock when they were received.

2. Correcting 3DM-GX1 and 3DM-GX3 Timing

Early test results showed that there was some amount of timing error between the time-stamped MicroStrain data and the time-stamped encoder data. This was identified by observing the sinusoidal values from both sources. There was a noticeable sinusoidal

error when differencing the two, which indicated a possible timing error. An example of the raw data that had a timing error is shown in Figure 45. Clearly, both the encoder and the sensor are tracking the same motion, but the time alignment introduced an artificial error, seen by the red dots of the 3DM-GX1 data appearing to lag behind the blue dots of the encoder data.

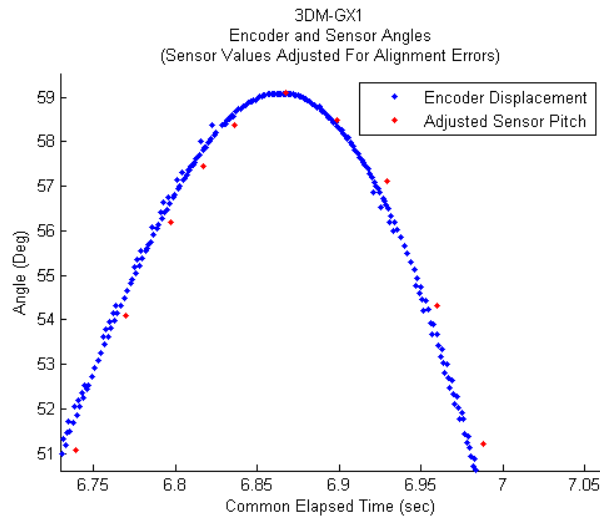


Figure 45. Encoder and Sensor Data, Detail Showing Timing Offset.

The difference in time between the two sets of data could be rectified by lining up the rising and falling lines of data and then identifying the offset value and applying that to all data times. To do this, peaks were identified in both the encoder data and the MicroStrain data. Since the MicroStrain data sampling period was not nearly as fast as the encoder, the exact peak was not always captured by the MicroStrain. To help compensate for this limitation, tests were always run such that they captured some amount of sinusoidal motion. When the test had many oscillations, multiple peaks were identified from the MicroStrain data. Some of the peaks occurred before the true peak, whereas others occurred just after the actual peak. Therefore, by averaging the difference between the identified peaks in the MicroStrain data and the encoder data, a reasonable estimate of the time-stamp error could be made and adjusted for.

In Figure 46, the error of the data in Figure 45 is plotted before any time adjustment was made, and the sinusoidal error can be clearly seen. In Figure 47, the same

data are used to calculate the error, but the times have been adjusted. With the method described previously, an 8.39 ms timing difference was removed. Notice that a significant amount of the error goes away, though there is still a slight sinusoidal characteristic to the error, indicating that there may still be some timing inaccuracies.

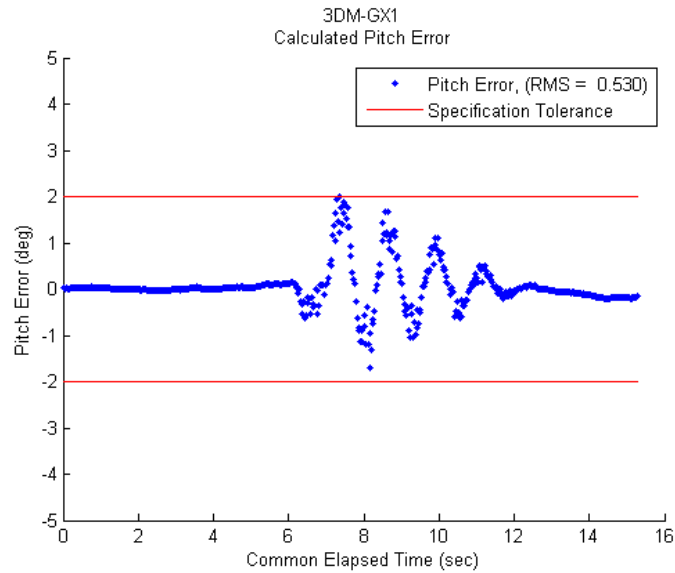


Figure 46. Error Plot With No Timing Correction.

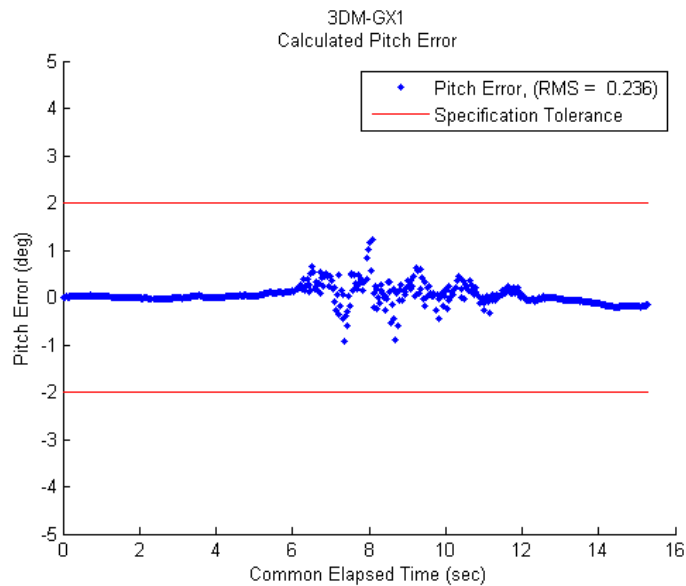


Figure 47. Error Plot With Timing Correction.

The point was not to make the error go to zero, but rather to get an accurate reading of the true error as output by the sensor. The details of this function are shown in Appendix B under the function “sensor_time_align.m.”

In order to ensure that the timing data could be aligned, there needed to be some amount of free swinging or sinusoidal data collected each test. This requirement is reflected in the methodology shown in Chapter VI. If the user did not want to adjust the time in this manner, then this feature could be turned off and the time adjustment could be manually input by the user with the MATLAB graphical user interface described in the next section.

G. MATLAB DATA PROCESSING

MATLAB R2008b was used for all data processing because of the ability to easily and rapidly manipulate data and because of the ease of which a user interface could be created for future data analysis. Each test that was run with the LabVIEW data collection and test apparatus created two data text files, one for the encoder and one for the sensor. These files can be read in by MATLAB functions and manipulated in order to align the times, remove excess data, and perform accuracy analysis. Standard plots are easily generated and data can be saved in processed form for future analysis. The following sections detail what was done by different functions and also introduces the GUI and explains how to use it to analyze data. The detailed commented code for all of the MATLAB functions used may be found in Appendix B.

1. Turning Raw Data Into Results

The basic flow of data, from raw data to graphs for easy analysis, proceeded in the following manner. The user provided the locations of the two data files to be analyzed and what type of sensor data it was, such as the 3DM-GX1 stabilized quaternion and vectors. Then the MATLAB functions read in the data and manipulated it appropriately, reshaping as necessary to have the first column represent time, and then each following column representing data that were collected. The encoder data were reshaped and aligned according to each element’s tick-time, all duplicate data values were removed,

then the remaining data were passed to the time alignment function described earlier. The MicroStrain data was also aligned in time order and adjusted, based on the previously described timing functions. A common elapsed time based on the CompactRIO time-stamp was calculated for both sets of data. The encoder data were passed to a function to calculate the “truth” data for angular velocity and acceleration.

It was assumed that for all tests the pendulum began in a steady, unmoving state. The data from the encoder and the MicroStrain sensor were averaged over the first two to three seconds, depending on the user selection. This value provided an “initial” value that was subtracted from all future values. Thus, all angles measured and displayed were relative to the starting angle.

Depending on the type of data that were pulled from the MicroStrain sensors, we generated certain variables containing data such as the time, encoder angle, the sensor roll, pitch, and yaw, angular rate data and more to be used for analysis. From these variables, data could be plotted and an accuracy analysis could be completed.

2. The Graphical User Interface “MEMS_Test”

A graphical user interface (GUI) was designed in order to speed up the data analysis and to enable easy future use. The GUI was created with future expandability in mind, such that the basic functionality required for the data collected in this thesis were not the only capabilities designed into the GUI. Many placeholders exist for upgrades and future capabilities, and some capabilities may be seen by the user but are not allowed to be enabled. The GUI does a certain amount of error checking to ensure that the data required for a certain action are available, though it was not fully vetted to be completely “user-proof.” The GUI was designed to enable simple, rapid data analysis such that a user only had to make a few selections and hit the “Run” button for all of the required functions to turn raw data into results. The window that first appears when running the GUI “MEMS_Test.m” is shown in Figure 48.

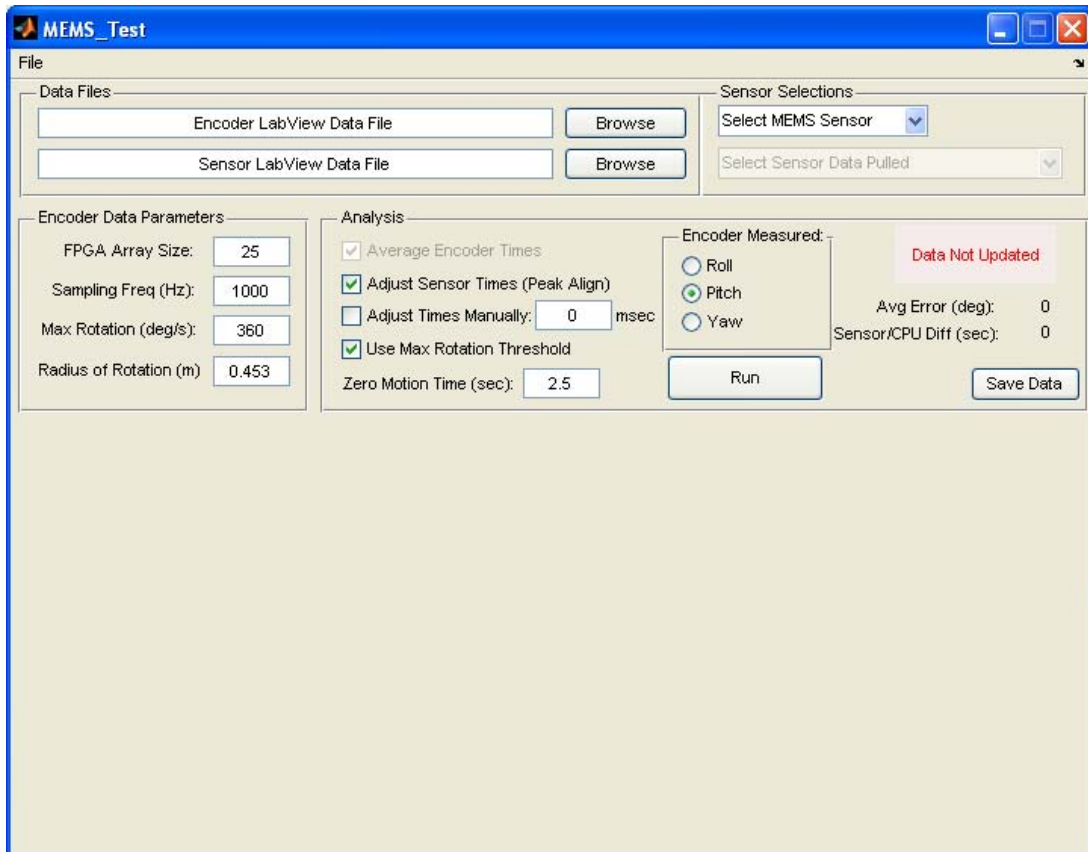


Figure 48. MEMS_Test GUI, Startup Window.

The user may browse for the encoder data text file and the sensor data text file, or those fields may be edited directly. The 3DM-GX1 or 3DM-GX3 sensors may be selected in the upper right, and the sensor data pulled matches what is expected based on the default LabVIEW data pulled shown in Section C above. The user may select another type of data from the sensor; however, at this time only the default data types are supported.

The box labeled “Encoder Data Parameters” contains a number of preset values required by some of the processing functions. The FPGA Array size should match the number of encoder values buffered on the FPGA, and the sampling frequency should match that in “encoder.VI” The Maximum Rotation rate may be used to threshold out any spurious points that are outside of these bounds, if desired.

The radius of rotation is used as r for (16) and (17) and is measured from the center of the pendulum axis of rotation to the center of the device under test.

The box labeled “Analysis” allows the user to choose to threshold the data according to the set value and to have the sensor data aligned as described in Section F or to manually enter a sensor time/CPU time adjustment. The “zero motion time” is the amount of time the pendulum was at rest while collecting data before the dynamic test began. The “Encoder Measured” radio buttons can be adjusted to indicate what values the encoder was measuring as truth.

The run button may be pressed to gather and compute all of the required data according to the setup parameters. When it has completed successfully a sound is played, and the indicator at the right turns green, indicating that the data have been updated. The average error is displayed and the time difference between the sensor data and the encoder data that was removed is displayed. If the “Adjust Sensor Times” button was not checked, then this will be blank.

The MATLAB variables read in and used for analysis data may be saved by pressing the save button. All of the pertinent variables in memory are saved to a “.mat” file in the same location as the encoder data, using the same title, with “Analyzed Data #.mat” replacing the “*.txt” at the end. The same encoder file may have up to nine data files stored with different parameters. Once the data are saved, the save button is disabled until changes are made to prevent unnecessary duplicate files.

Any time the critical parameters used to make computations are changed in the GUI, the “Data Not Updated” indicator goes red and indicates that the current values displayed on the GUI may not match what is in memory. Although data may still be used for plotting, saving is disabled

After successfully running and storing variables in memory, the plotting half of the GUI is revealed, as shown in Figure 49.

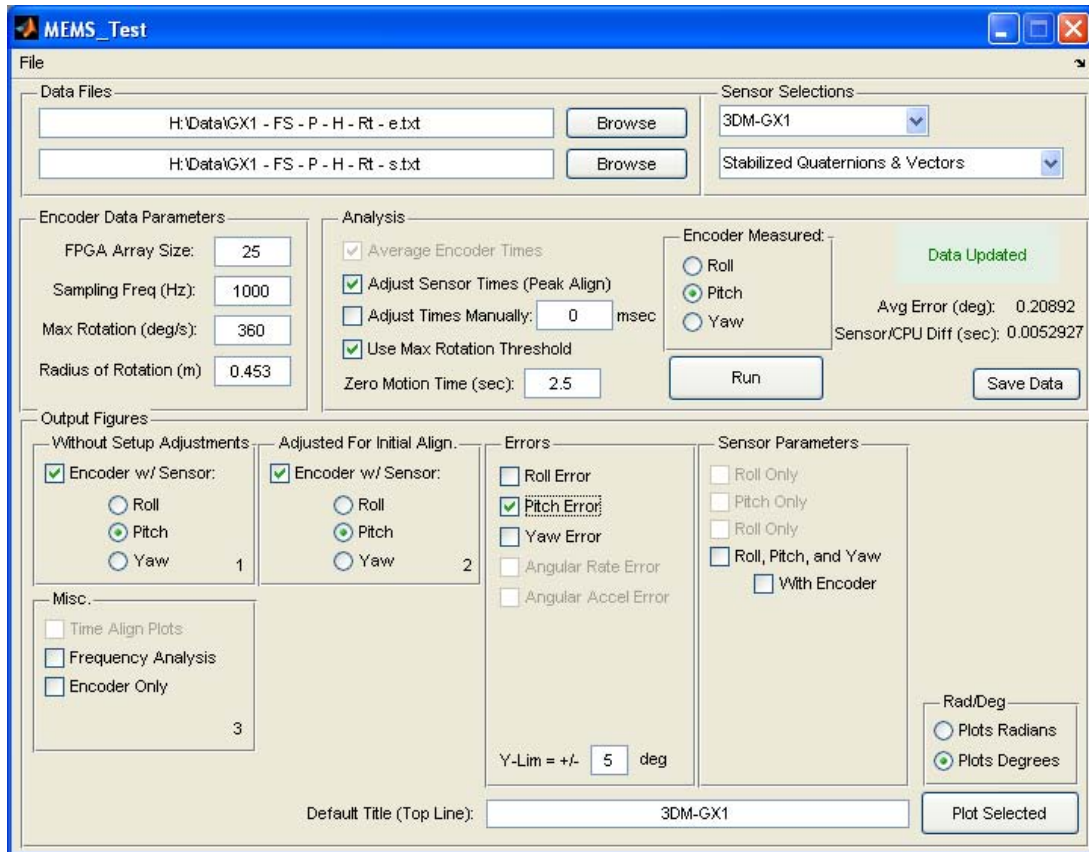


Figure 49. MEMS_Test GUI, Plotting Window Revealed.

The desired output plots may be selected at any time, and when all selections have been made the “Plot Selected” button creates a separate figure for each checked item. Plots may be shown in radians or degrees, but this selection must be made before hitting the plot button. The text in the “Default Title” line is plotted as the top line of the title in all of the figures and defaults to the sensor under test, though the text may be changed by the user as desired.

The plot “Without Setup Adjustments” plots the data without the sensor adjusted to have the average value of first few seconds equal zero degrees. In Figure 50, two plots of the same data are shown. The one on the left was plotted without setup adjustments, and the one on the right was adjusted for initial alignment errors. Note that the error plots are always computed with the data that have been adjusted for initial alignment, regardless of whether or not that box is checked.

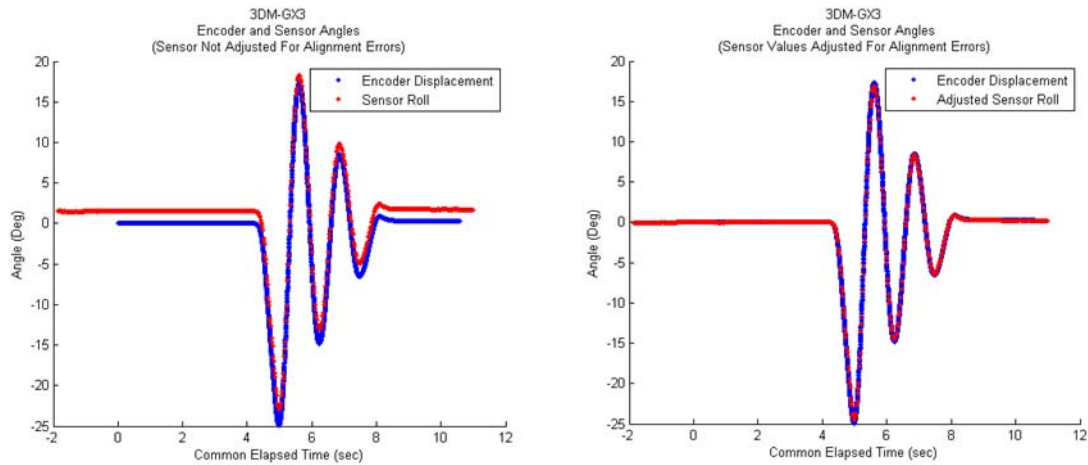


Figure 50. Sample Sensor Plots Without and With Adjustments, Respectively.

Many plots of the errors as well as the Roll, Pitch, and Yaw plotted together are shown in Chapters VI and VII and are not shown here. The FFT plots shown in this chapter, Section E, were generated by checking the “Frequency Analysis” button. All data figures shown in this thesis were generated using the MEMS_Test GUI.

H. SUMMARY

In this chapter, the specifics of the data collection scheme used in this thesis were provided. The National Instruments hardware and the LabVIEW virtual instruments used to gather the data were shown and explained. The use of the encoder as the truth source was discussed and limitations were presented for deriving angular velocity and angular acceleration. The timing of the data samples was discussed and a MATLAB GUI was presented for quick and simple data analysis.

In the next chapter, the specific test methodology of the five tests performed on the 3DM-GX1 and 3DM-GX3 is introduced. Test procedures are provided showing exactly how to gather the data and samples are shown of the encoder data collected on each test. In the end, a test matrix is put forth showing all of the tests performed for this thesis.

VI. TEST METHODOLOGY

In this chapter, the different types of tests performed are explained. A detailed description is given of each test. For every test, it is assumed that the data collection equipment was set up according to Section C.5 of Chapter V. For the tests involving impact, the modified test apparatus with the impact arm introduced in Chapter IV was used. For each type of test, representative plots of the encoder data are provided to show the types of motion traced by each test. Finally, a test matrix is provided showing what tests were executed and the type of data collect from each to be analyzed, either roll, pitch, or yaw.

A. FREE SWINGING

The “free swinging” tests simply gathered the motion of the free swinging pendulum. When the test apparatus was vertical, the pendulum was pulled to the low, medium, or high mark on the reference board and immediately released and allowed to swing freely. The mount bearings had some amount of resistance and provided damping for the pendulum. Once the pendulum had stopped, data collection was ceased. Data were to be collected for each case, low/med/high, starting with the pendulum drawn initially to the right and also to the left. When the test apparatus was horizontal to measure yaw, a metronome was used to mark time at one beat per second. Using the metronome as an aid, we moved the pendulum between the low, med, or high marks and attempted to swing from one mark to the other in one second, before gradually damping out similar to the vertical case. A comparable amount of data were collected as the vertical case, and then the test was stopped.

The steps for conducting this test were as follows.

- Begin at neutral (center mark) *before* data collection was started.
- Start data collection. Do not move the pendulum for 2–3 seconds.
- If Pendulum Vertical: Move the pendulum either to the low, medium, or high mark and immediately release.

- If Pendulum Horizontal: Start metronome. Move the pendulum between the low, medium, or high mark, attempting to hit the mark at every beat of the metronome. Damp out to initial point.
- When the pendulum had stopped (or after a few cycles in the horizontal test case), cease data collection.

Representative plots of the encoder data collected for low, medium, and high tests are shown in Figures 51, 52, and 53, respectively. The angle directly from the encoder is shown in blue, and the interpolated values that were used to calculate the error between the encoder and the sensor are shown in green.

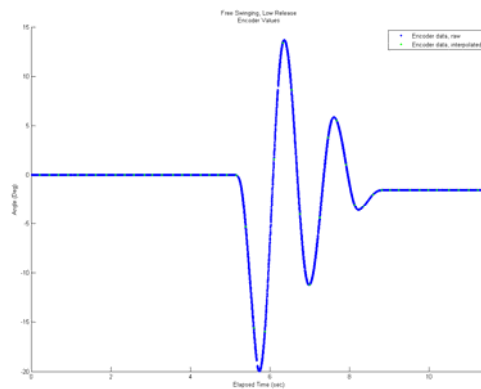


Figure 51. Free Swinging, Representative Encoder Angle, Low Release.

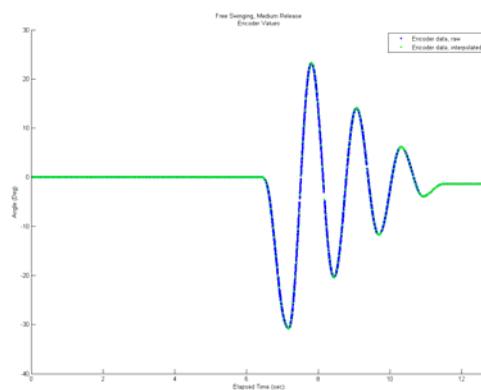


Figure 52. Free Swinging, Representative Encoder Angle, Medium Release.

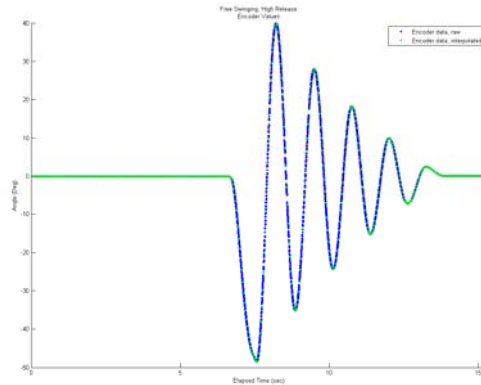


Figure 53. Free Swinging, Representative Encoder Angle, High Release.

B. ARBITRARY SWINGING

The “arbitrary swinging” tests were similar to the free swinging tests, except instead of drawing back and letting it go, the pendulum was moved in an arbitrary fashion either at a slow speed or a fast speed. When moving the pendulum in an arbitrary way a “slow” speed of movement was maintained by using a metronome to count off one beat per second, ensuring that no more than one mark on the reference board was passed in a second. A “fast” speed indicated that one mark or more were passed per second. As much as possible, a chaotic motion was used in both the horizontal and vertical test cases. A duration of 10 to 15 seconds of data were collected for roll, pitch, and yaw in fast and slow motions.

The steps for conducting this test were as follows.

- Begin at neutral (center mark) *before* data collection was started.
- Start data collection and metronome. Do not move the pendulum for 2–3 seconds.
- Begin moving pendulum in arbitrary/chaotic fashion for 10–15 seconds.
- Cease data collection.

Representative plots of the encoder data collected for slow and fast tests are shown in Figures 54 and 55, respectively. The angle directly from the encoder is shown in blue, and the interpolated values that were used to calculate the error between the encoder and the sensor are shown in green.

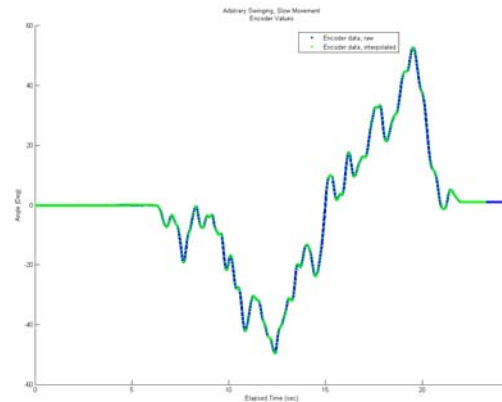


Figure 54. Arbitrary Swinging, Representative Encoder Angle, Slow Movement.

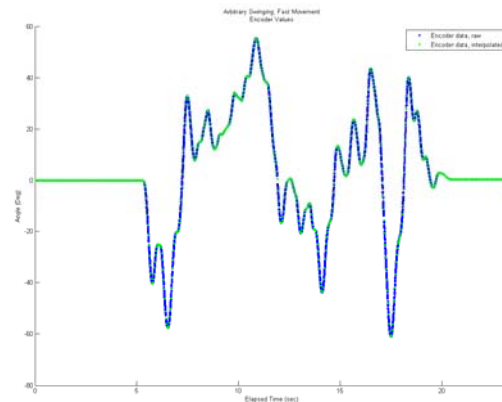


Figure 55. Arbitrary Swinging, Representative Encoder Angle, Fast Movement.

C. SEMI-STATIC: MOVE AND HOLD

The “semi-static, move and hold” tests examined the static response immediately following dynamic motion and visa versa. The pendulum began at neutral and then was moved to the point of maximum deflection and held for approximately 4 seconds. It was then released and allowed to swing freely for a couple of cycles, and then the data

collection was ceased. The pendulum was moved to maximum deflection both slowly, taking at least 3 seconds to get to the maximum value, and quickly, taking only 1 second or less to get to the maximum deflection. Movements to both the right and left sides of the reference board were tested.

The steps for conducting this test were as follows.

- Begin at neutral (center mark) *before* data collection was started.
- Start data collection and metronome. Do not move the pendulum for 2–3 seconds.
- Using the slow or fast speeds defined above, move pendulum smoothly to maximum deflection.
- When pendulum hits the cantilevered arm, hold it there for ~4 seconds.
- Release pendulum and allow free swinging for a couple of cycles.
- Cease data collection.

Representative plots of the encoder data collected for slow and fast tests are shown in Figures 56 and 57, respectively. The angle directly from the encoder is shown in blue, and the interpolated values that were used to calculate the error between the encoder and the sensor are shown in green.

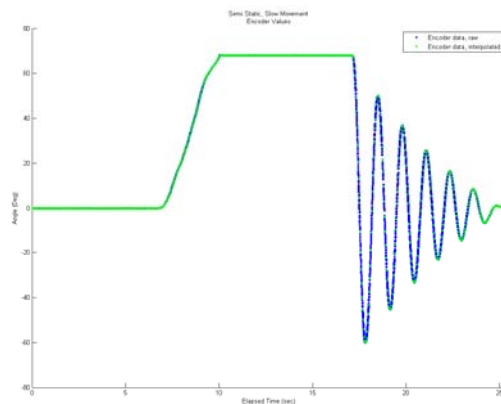


Figure 56. Semi-Static, Representative Encoder Angle, Slow Movement.

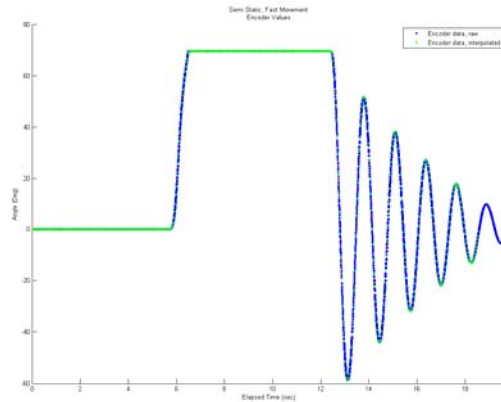


Figure 57. Semi-Static, Representative Encoder Angle, Fast Movement.

D. FREE SWING WITH IMPACT

In order to run the “free swing with impact” tests, the test apparatus had to be modified in accordance with Section F.1 of Chapter IV. The “free swing with impact” tests were run in a very similar manner as the free swinging pendulum, in that the pendulum was drawn back to the low, medium, or high mark and then immediately released. It was allowed to swing freely and impact the board, bounce off, and then damp out to neutral. In the horizontal configuration, the pendulum was drawn back and then swung forward into the board and allowed to bounce off and return back to rest.

The steps for conducting this test were as follows.

- Begin at neutral (center mark) *before* data collection was started.
- Start data collection. Do not move the pendulum for 2–3 seconds.
- If Pendulum Vertical: Draw the pendulum either to the low, medium, or high mark and release.
- If Pendulum Horizontal: Move the pendulum to the low, medium, or high mark, and then swing the pendulum toward the board.
- Allow the pendulum to impact and bounce off of the board, eventually coming to rest.
- Cease data collection.

Representative plots of the encoder data collected for low, medium, and high tests are shown in Figures 58, 59, and 60, respectively. The angle directly from the encoder is shown in blue, and the interpolated values that were used to calculate the error between the encoder and the sensor are shown in green.

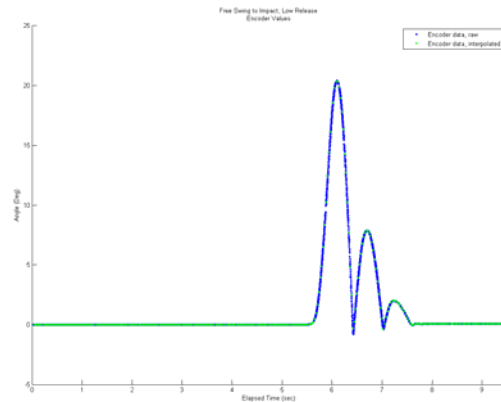


Figure 58. Free Swing to Impact, Representative Encoder Angle, Low Release.

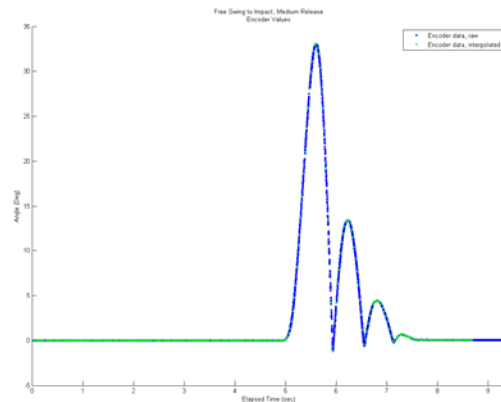


Figure 59. Free Swing to Impact, Representative Encoder Angle, Medium Release.

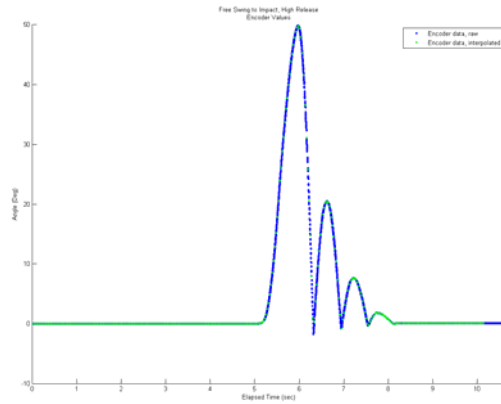


Figure 60. Free Swing to Impact, Representative Encoder Angle, High Release.

E. FREE SWING WITH IMPACT AND HOLD

The “free swing with impact and hold” was nearly identical to the “free swing with impact” tests, except that instead of allowing the pendulum to bounce off of the impact board, a piece of Velcro was used to have the pendulum stick to the impact board and hold in place. Since the Velcro still allowed a very small amount of “bounce back,” the pendulum was manually pressed to the impact arm and held immediately after impact.

The steps for conducting this test were as follows.

- Begin at neutral (center mark) *before* data collection was started.
- Start data collection. Do not move the pendulum for 2–3 seconds.
- If Pendulum Vertical: Draw the pendulum either to the low, medium, or high mark and release.
- If Pendulum Horizontal: Move the pendulum to the low, medium, or high mark, and then swing the pendulum toward the board.
- Allow the pendulum to impact and stick for ~4 seconds.
- Cease data collection.

Representative plots of the encoder data collected for low, medium, and high tests are shown in Figures 61, 62, and 63, respectively. The angle directly from the encoder is

shown in blue, and the interpolated values that were used to calculate the error between the encoder and the sensor are shown in green.

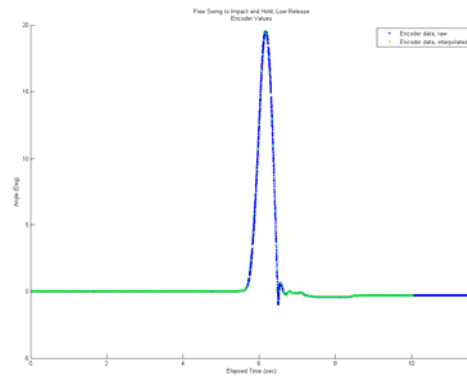


Figure 61. Impact and Hold, Representative Encoder Angle, Low Release.

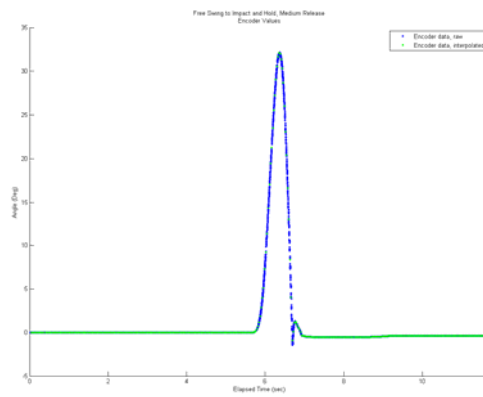


Figure 62. Impact and Hold, Representative Encoder Angle, Medium Release.

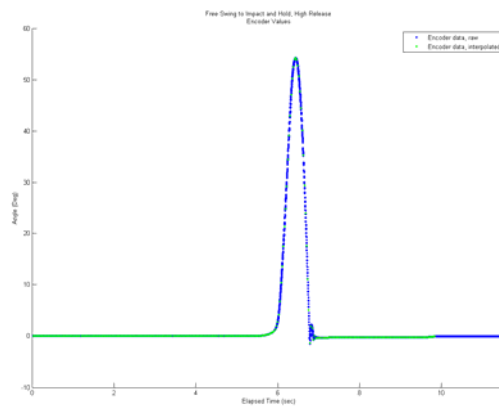


Figure 63. Impact and Hold, Representative Encoder Angle, High Release.

F. TEST MATRIX

A test matrix was created to ensure all of the data were collected. One matrix was built for all of the tests not involving any impact and is shown in Table 6. Another test matrix was built for all tests involving an impact and is shown in Table 7.

Table 6. Non-Impact Test Matrix.

Free Swinging (Roll)	3DM-GX1		3DM-GX3	
	Start Left	Start Right	Start Left	Start Right
Low				
Medium				
High				

Arbitrarily Swinging (Roll)	3DM-GX1	3DM-GX3
Slow		
Fast		

Semi-Static (Roll)	3DM-GX1		3DM-GX3	
	Move Left	Move Right	Move Left	Move Right
Slow				
Fast				

Free Swinging (Pitch)	3DM-GX1		3DM-GX3	
	Start Left	Start Right	Start Left	Start Right
Low				
Medium				
High				

Arbitrarily Swinging (Pitch)	3DM-GX1	3DM-GX3
Slow		
Fast		

Semi-Static (Pitch)	3DM-GX1		3DM-GX3	
	Move Left	Move Right	Move Left	Move Right
Slow				
Fast				

Free Swinging (Yaw)	3DM-GX1		3DM-GX3	
	Start Left	Start Right	Start Left	Start Right
Low				
Medium				
High				

Arbitrarily Swinging (Yaw)	3DM-GX1	3DM-GX3
Slow		
Fast		

Semi-Static (Yaw)	3DM-GX1		3DM-GX3	
	Move Left	Move Right	Move Left	Move Right
Slow				
Fast				

Table 7. Impact Test Matrix.

Free Swinging To Impact: (Roll)	3DM-GX1	3DM-GX3
Low		
Medium		
High		

Free Swinging To Impact & Hold: (Roll)	3DM-GX1	3DM-GX3
Low		
Medium		
High		

Free Swinging To Impact: (Pitch)	3DM-GX1	3DM-GX3
Low		
Medium		
High		

Free Swinging To Impact & Hold: (Pitch)	3DM-GX1	3DM-GX3
Low		
Medium		
High		

Free Swinging To Impact: (Yaw)	3DM-GX1	3DM-GX3
Low		
Medium		
High		

Free Swinging To Impact & Hold: (Yaw)	3DM-GX1	3DM-GX3
Low		
Medium		
High		

For each open box a single test was run at those conditions and the data were recorded for analysis. The error plots of the results are shown in Chapter VII.

G. SUMMARY

The step-by-step methodology used to test the MEMS sensors was provided in this chapter, and a test matrix showing all of the tests that were performed was laid out.

In the next chapter, the accuracy plots are presented for the two MicroStrain sensors examined for each of the tests in the provided matrix. For certain tests, additional plots are shown to provide added levels of detail. The goal is not to provide a comprehensive analysis of either sensor, but rather to show what types of tests can be performed and to collect some accuracy data and apply it to the sensors' potential use in either personal navigation or human motion tracking.

VII. RESULTS AND DATA PLOTS

In this chapter, the results of the testing accomplished with the new test apparatus are presented. First, the calculations for the error are defined. Then, for each test in the matrix introduced in Chapter VI an error plot is shown. For select tests the roll, pitch, and yaw are all plotted on one figure to show any “cross-talk.” Although a large amount of data are presented, the goal is not to conduct an exhaustive accuracy analysis survey on either sensor, but rather to show the types of test data that can be collected as well as to partially characterize each sensor with the specific application of a personal navigation system or human motion tracking in mind.

Whenever possible, the automatic sensor time align described in Chapter V was used when reading the data for this section. However, in some instances, the algorithm failed to adequately align the data, and for those cases an offset was manually calculated based on the best approximation from the unadjusted data.

A. ACCURACY DEFINITIONS

Based on Tables 1 and 2 from Chapter II, the dynamic accuracy for both the 3DM-GX1 and 3DM-GX3 should be ± 2 degrees. All accuracy plots shown in this chapter have lines at these values plotted for reference, though it is assumed that the accuracy specification is a root mean squared (RMS) accuracy and not an instantaneous accuracy. However, both the RMS and instantaneous accuracies are shown in all plots since in certain applications the instantaneous accuracy may be very important, whereas other applications may only be concerned with the RMS accuracy.

Since the encoder data was not valid at the exact times the sensor data was valid, the data from the encoder were interpolated in order to match the times of the sensor data. The difference between the sensor value and the encoder value interpolated for that sensor time made up the absolute error. The RMS of the collection of individual absolute errors was calculated by

$$Error_{RMS} = \sqrt{\text{mean}\left\{\left(s[n] - a[n]\right)^2\right\}} \quad (18)$$

for all n , where $s[n]$ was the sensor value and $a[n]$ was the interpolated angular value of the encoder.

Since this thesis was more concerned with the dynamic performance than with long term static and dynamic performance, the RMS error calculations shown only took into account the errors *after* the user entered zero motion time, typically 2.5 seconds.

B. 3DM-GX1

This section contains the accuracy results of the tests performed according to the methodology set out in Chapter VI for the 3DM-GX1. For each test the roll, pitch, and yaw error plots are shown.

1. Free Swinging

The following sections contain the accuracy results of the Free Swinging tests performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the Free Swinging tests for the 3DM-GX1 are shown in Figures 64, 65, and 66 for the low, medium, and high release test cases, respectively.

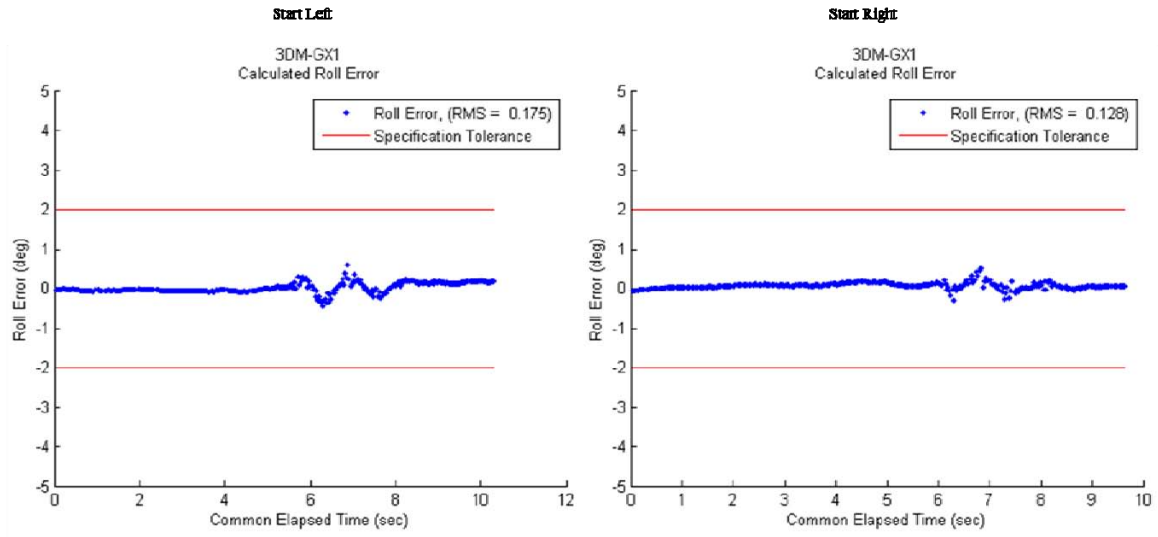


Figure 64. 3DM-GX1, Free Swinging Test, Roll Accuracy, Low Release Angle.

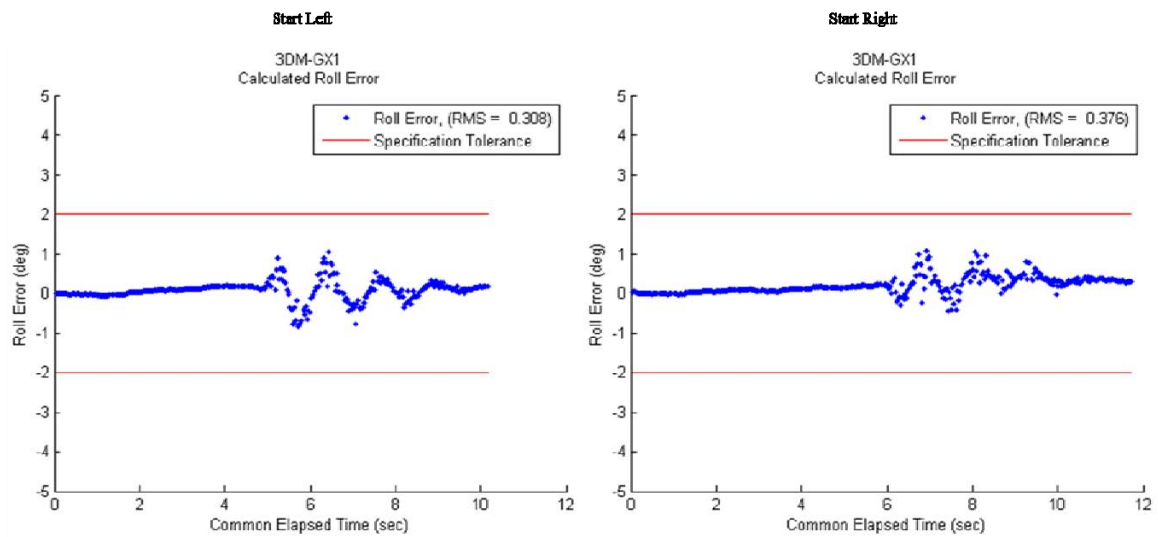


Figure 65. 3DM-GX1, Free Swinging Test, Roll Accuracy, Medium Release Angle.

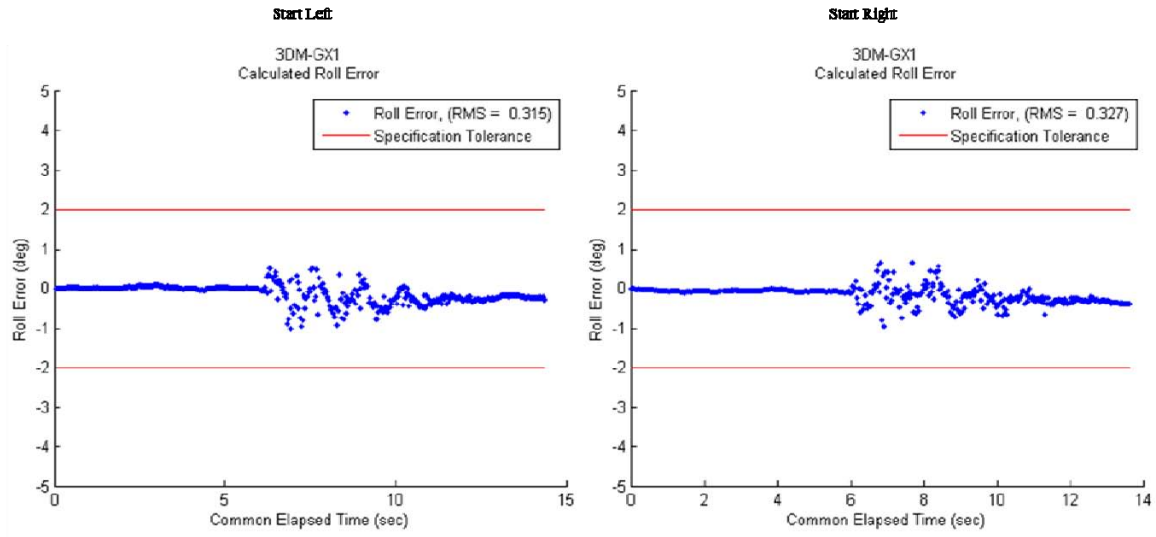


Figure 66. 3DM-GX1, Free Swinging Test, Roll Accuracy, High Release Angle.

b. Pitch

The pitch accuracy results of the Free Swinging tests for the 3DM-GX1 are shown in Figures 67, 68, and 69 for the low, medium, and high release test cases, respectively.

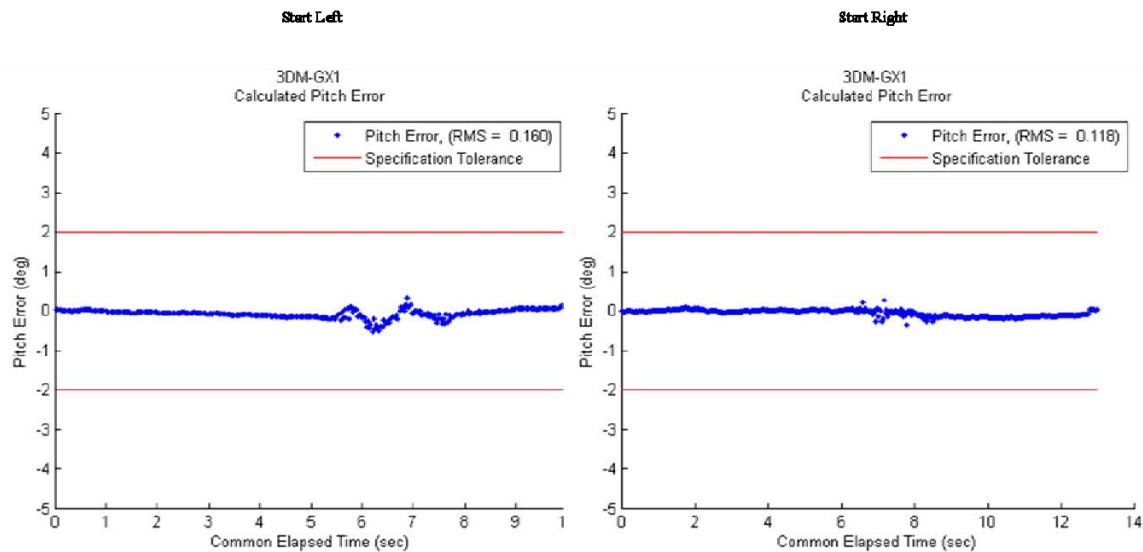


Figure 67. 3DM-GX1, Free Swinging Test, Pitch Accuracy, Low Release Angle.

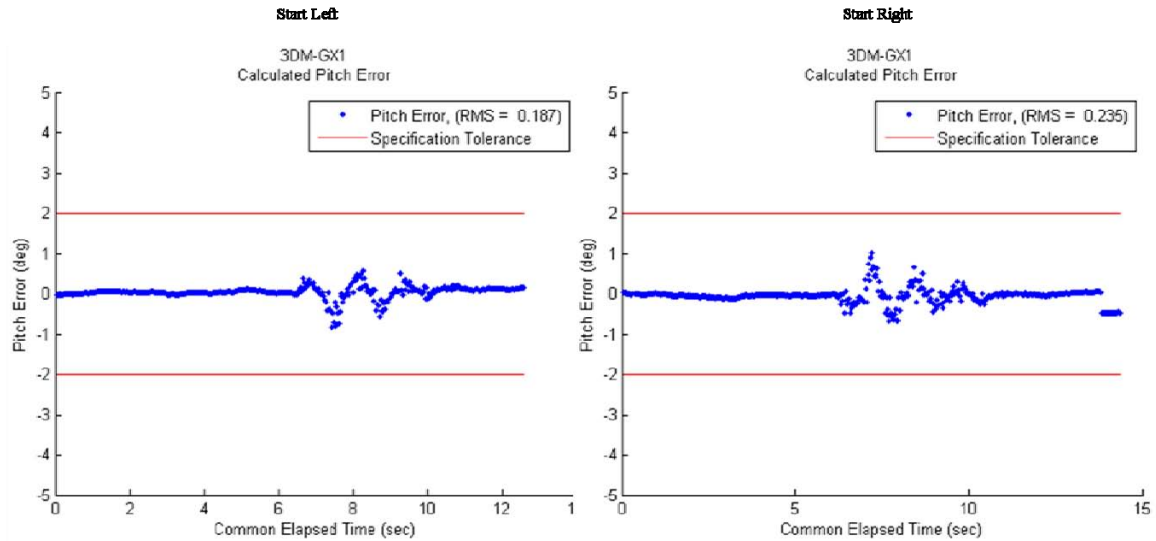


Figure 68. 3DM-GX1, Free Swinging Test, Pitch Accuracy, Medium Release Angle.

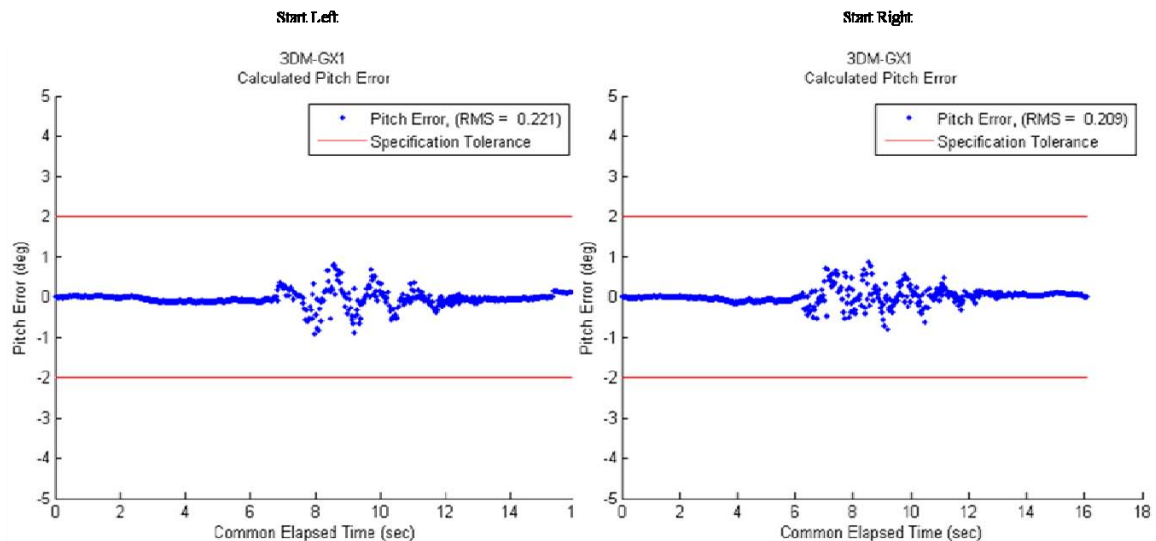


Figure 69. 3DM-GX1, Free Swinging Test, Pitch Accuracy, High Release Angle.

c. Yaw

The yaw accuracy results of the Free Swinging tests for the 3DM-GX1 are shown in Figures 70, 71, and 72 for the low, medium, and high release test cases, respectively. In Figure 72, there appears to be an induced drift that did not show up in any of the other free swinging yaw tests. The cause of the drift is unknown.

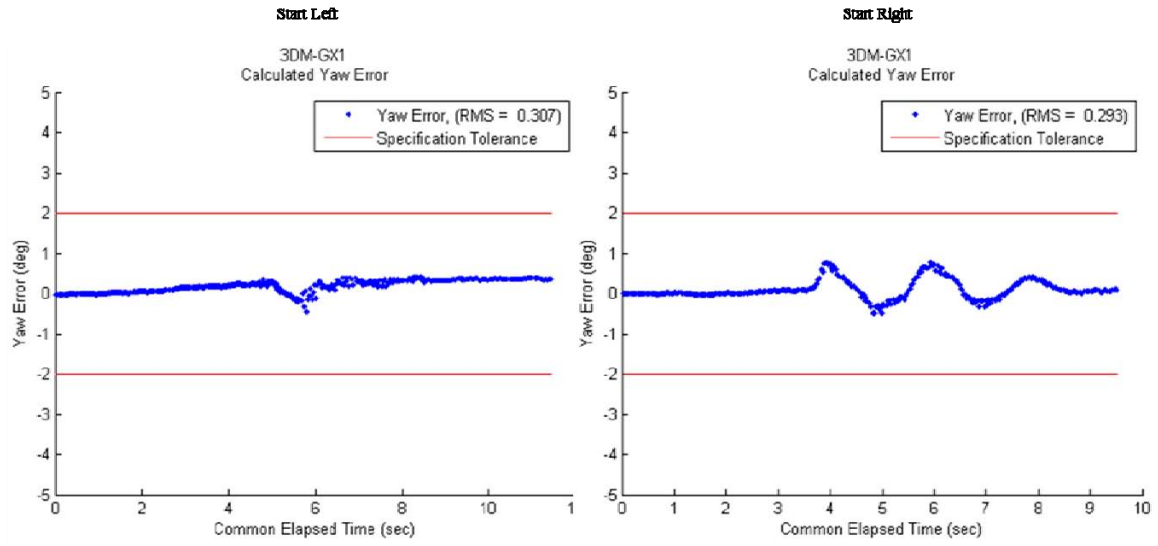


Figure 70. 3DM-GX1, Free Swinging Test, Yaw Accuracy, Low Release Angle.

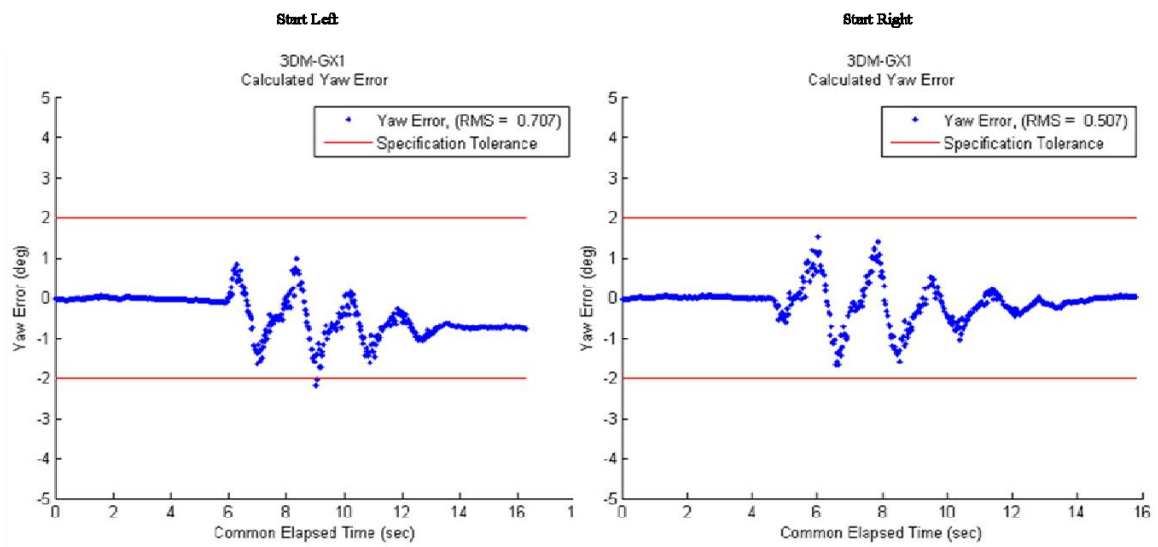


Figure 71. 3DM-GX1, Free Swinging Test, Yaw Accuracy, Medium Release Angle.

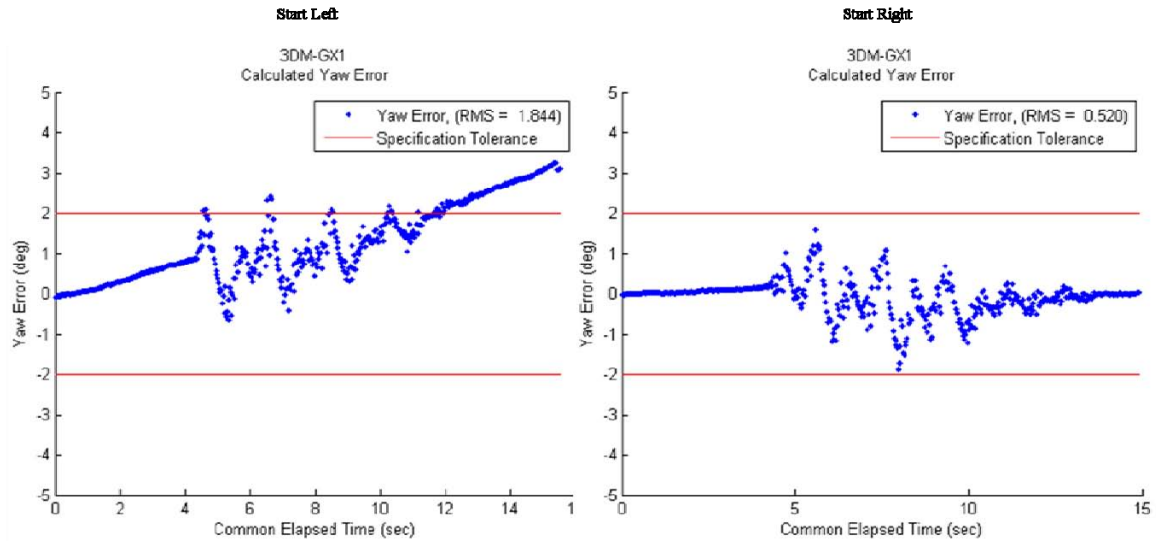


Figure 72. 3DM-GX1, Free Swinging Test, Yaw Accuracy, High Release Angle.

d. Free Swinging Tests' Cross-Talk

Although attempts were made to isolate each axis under test, some amount of cross-talk was seen. This was partly due to the test apparatus setup and partly due to errors in the sensor. In Figures 73, 74, and 75, the unadjusted roll, pitch, and yaw are all plotted on the same figure, one figure for each test apparatus configuration. The selected values for all three plots are shown for a “high” release case.

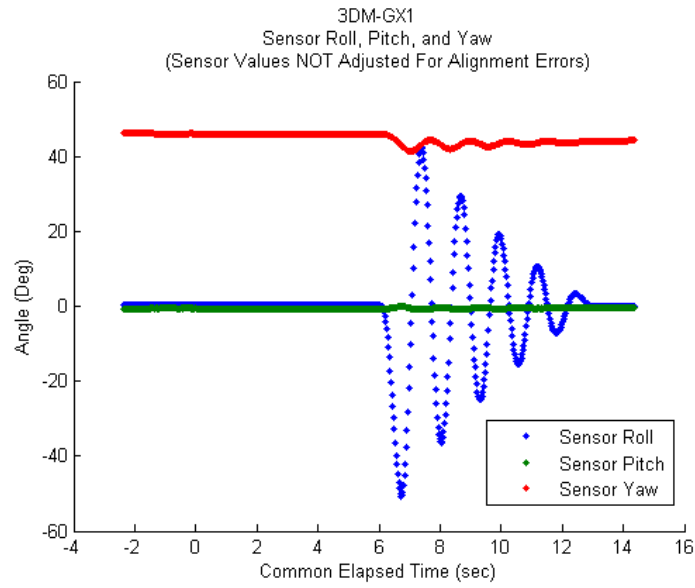


Figure 73. 3D-MGX1, Free Swinging Test, Cross-Talk Plot, Roll Under Test.

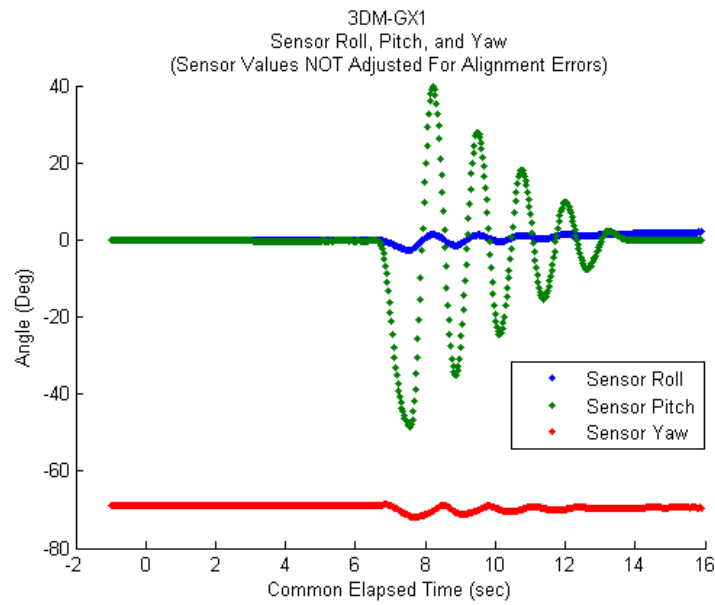


Figure 74. 3D-MGX1, Free Swinging Test, Cross-Talk Plot, Pitch Under Test.

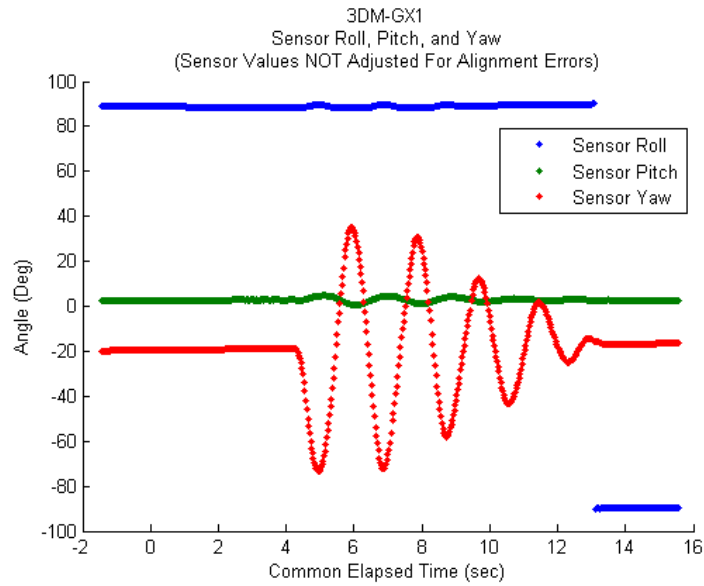


Figure 75. 3DM-GX1, Free Swinging Test, Cross-Talk Plot, Yaw Under Test.

2. Arbitrary Swinging

The following sections contain the accuracy results of the Arbitrary Swinging tests performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the Arbitrary Swinging tests for the 3DM-GX1 are shown in Figure 76. Both the slow and the fast movement accuracies are shown.

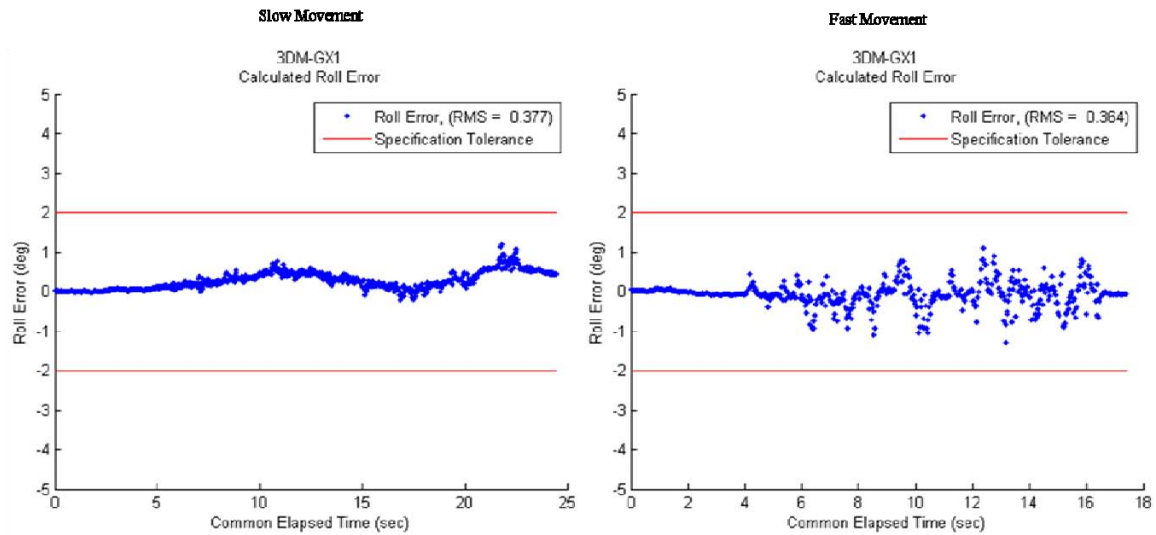


Figure 76. 3DM-GX1, Arbitrary Swinging Test, Roll Accuracy.

b. Pitch

The pitch accuracy results of the Arbitrary Swinging tests for the 3DM-GX1 are shown in Figure 77. Both the slow and the fast movement accuracies are shown.

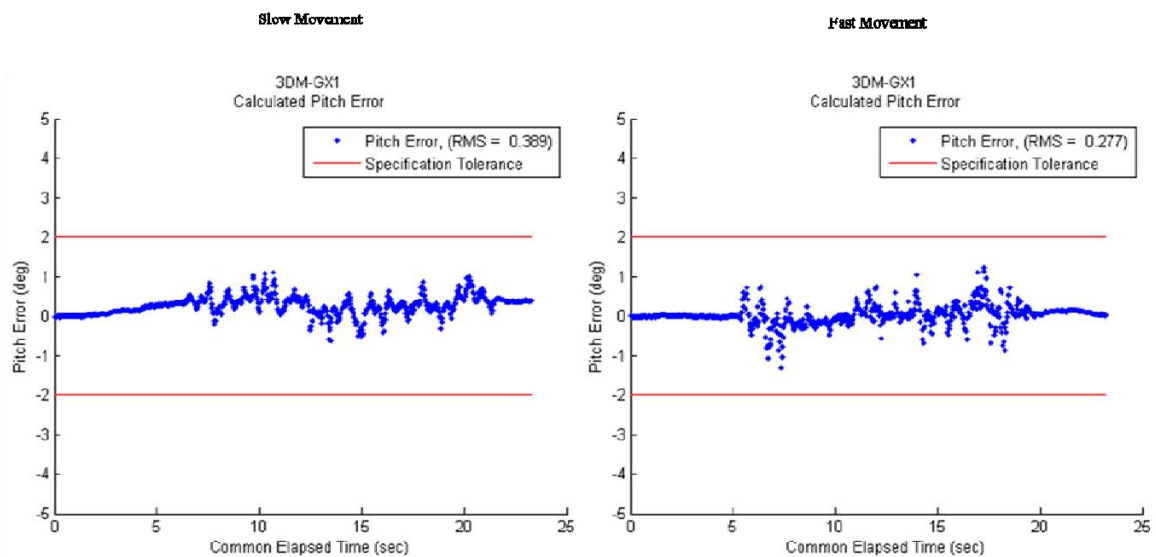


Figure 77. 3DM-GX1, Arbitrary Swinging Test, Pitch Accuracy.

c. Yaw

The yaw accuracy results of the Arbitrary Swinging tests for the 3DM-GX1 are shown in Figures 78. Both the slow and the fast movement accuracies are shown.

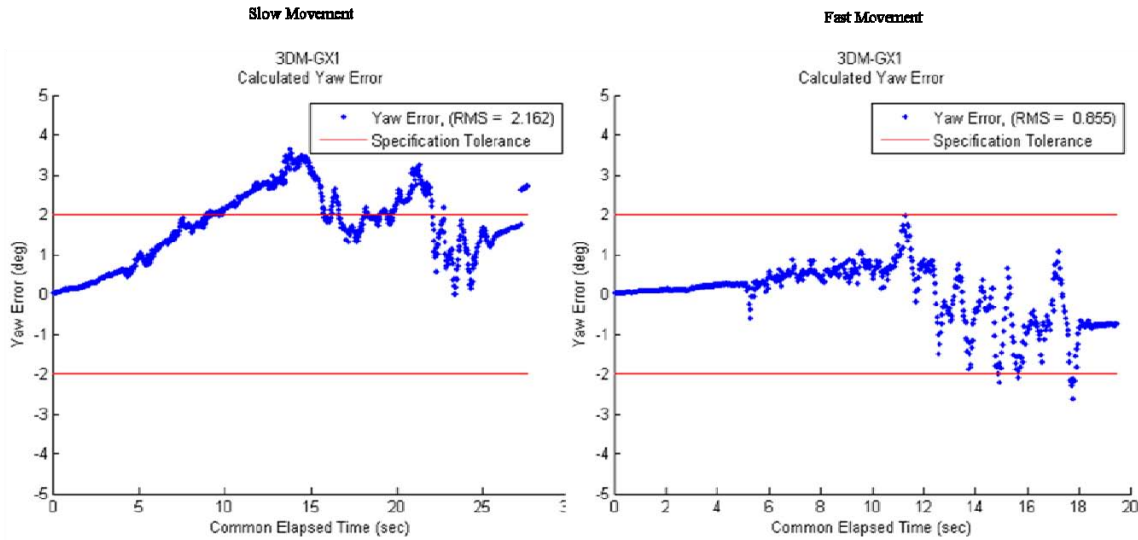


Figure 78. 3DM-GX1, Arbitrary Swinging Test, Yaw Accuracy.

3. Semi-Static: Move and Hold

The following sections contain the accuracy results of the Semi-Static tests performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the Semi-Static tests for the 3DM-GX1 are shown in Figure 79 for both the slow and fast cases starting to the left and to the right. In Figure 80, the point where maximum deflection is reached when moving quickly to the right is zoomed in to show in detail the behavior of the sensor at that point.

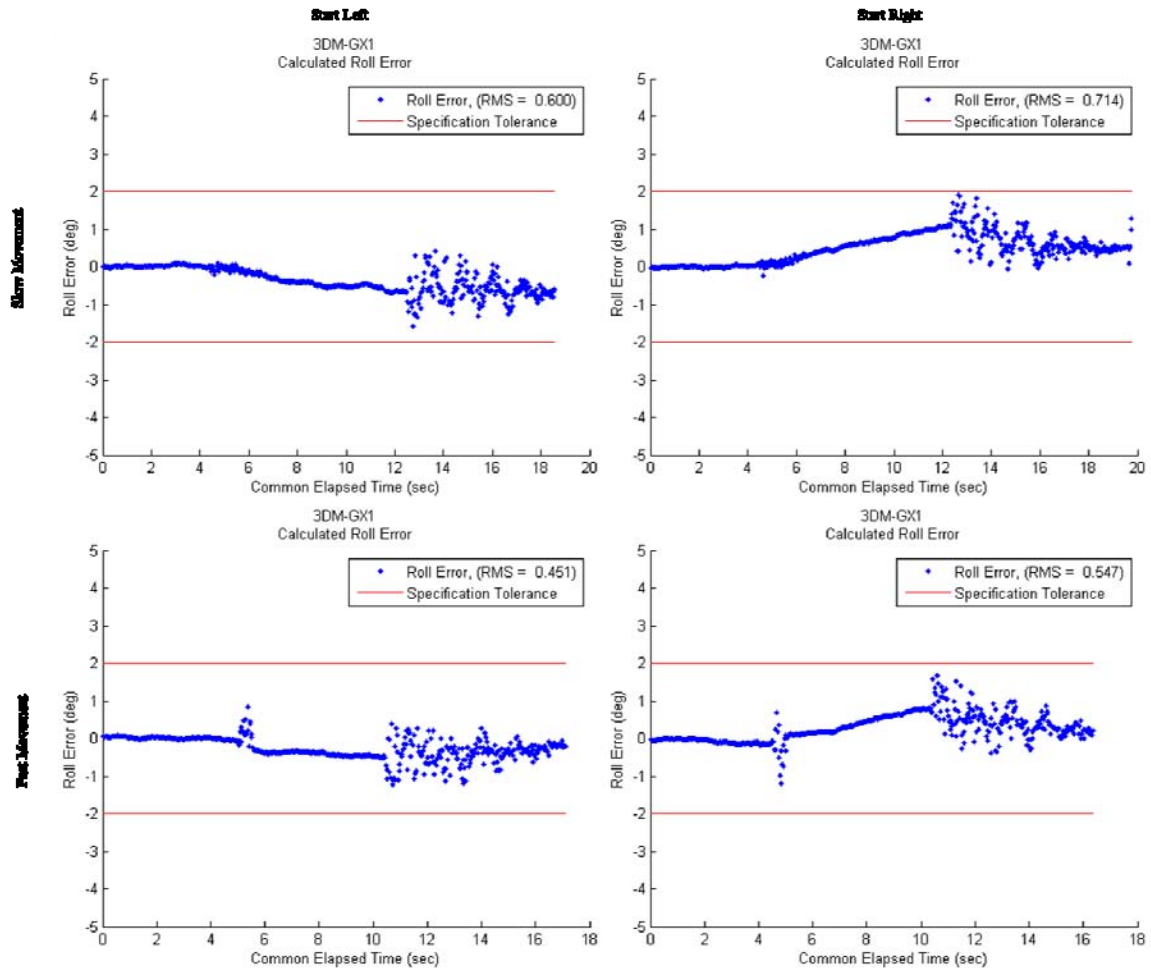


Figure 79. 3DM-GX1, Semi-Static Test, Roll Accuracy.

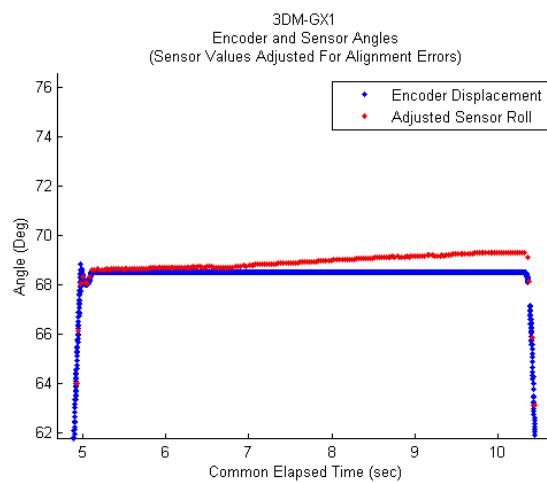


Figure 80. 3DM-GX1, Semi-Static Test, Roll Accuracy, Zoomed in to Show Detail.

b. Pitch

The pitch accuracy results of the Semi-Static tests for the 3DM-GX1 are shown in Figure 81 for both the slow and fast cases starting to the left and to the right. In Figure 82, the point where maximum deflection is reached when moving quickly to the right is zoomed in to show in detail the behavior of the sensor at that point.

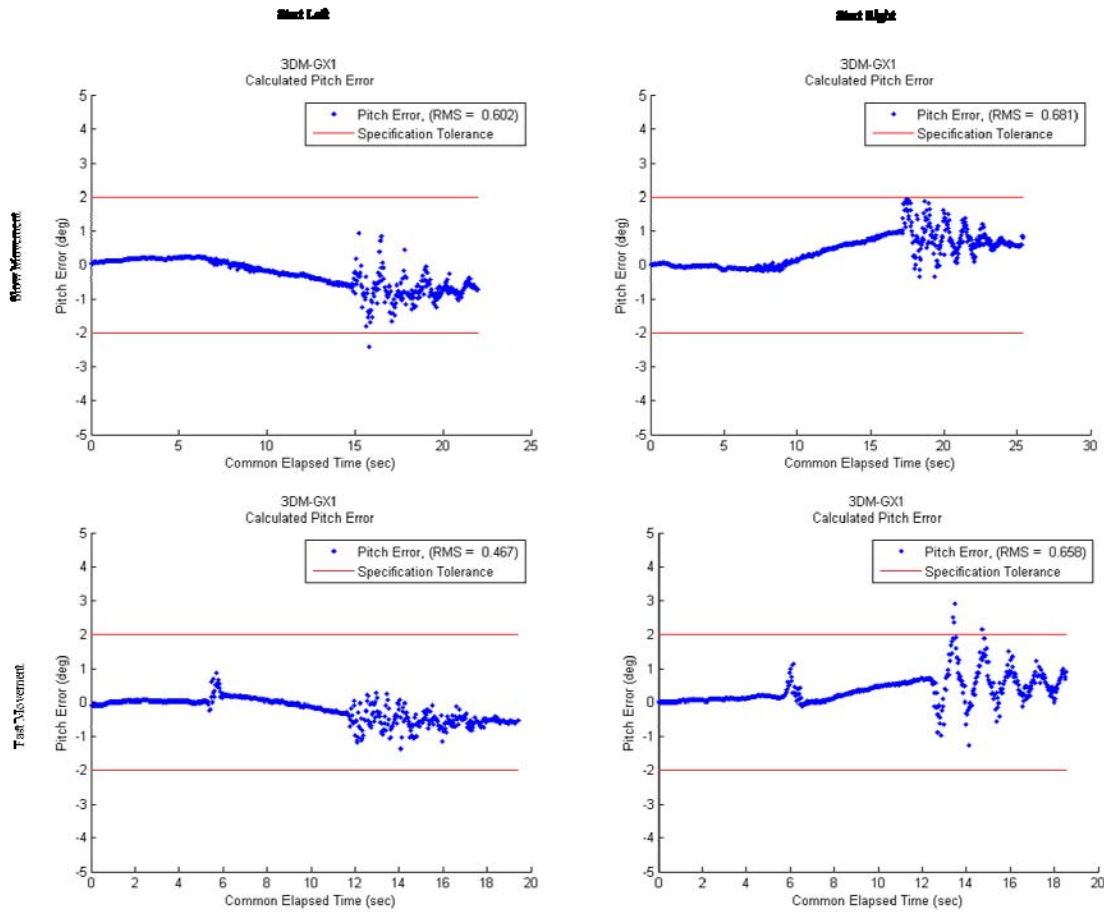


Figure 81. 3DM-GX1, Semi-Static Test, Pitch Accuracy.

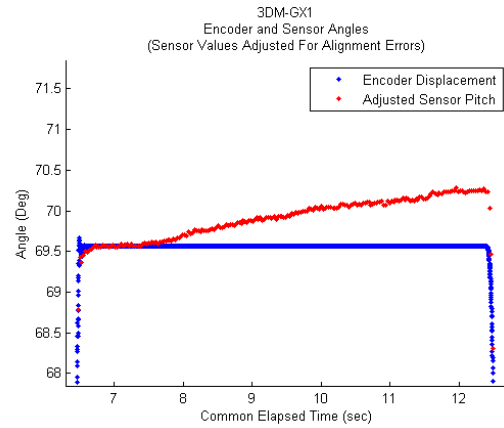


Figure 82. 3DM-GX1, Semi-Static Test, Pitch Accuracy, Zoomed in to Show Detail.

c. Yaw

The yaw accuracy results of the Semi-Static tests for the 3DM-GX1 are shown in Figure 83 for both the slow and fast cases starting to the left and to the right. In Figure 84, the point where maximum deflection is reached when moving quickly to the left is zoomed in to show in detail the behavior of the sensor at that point.

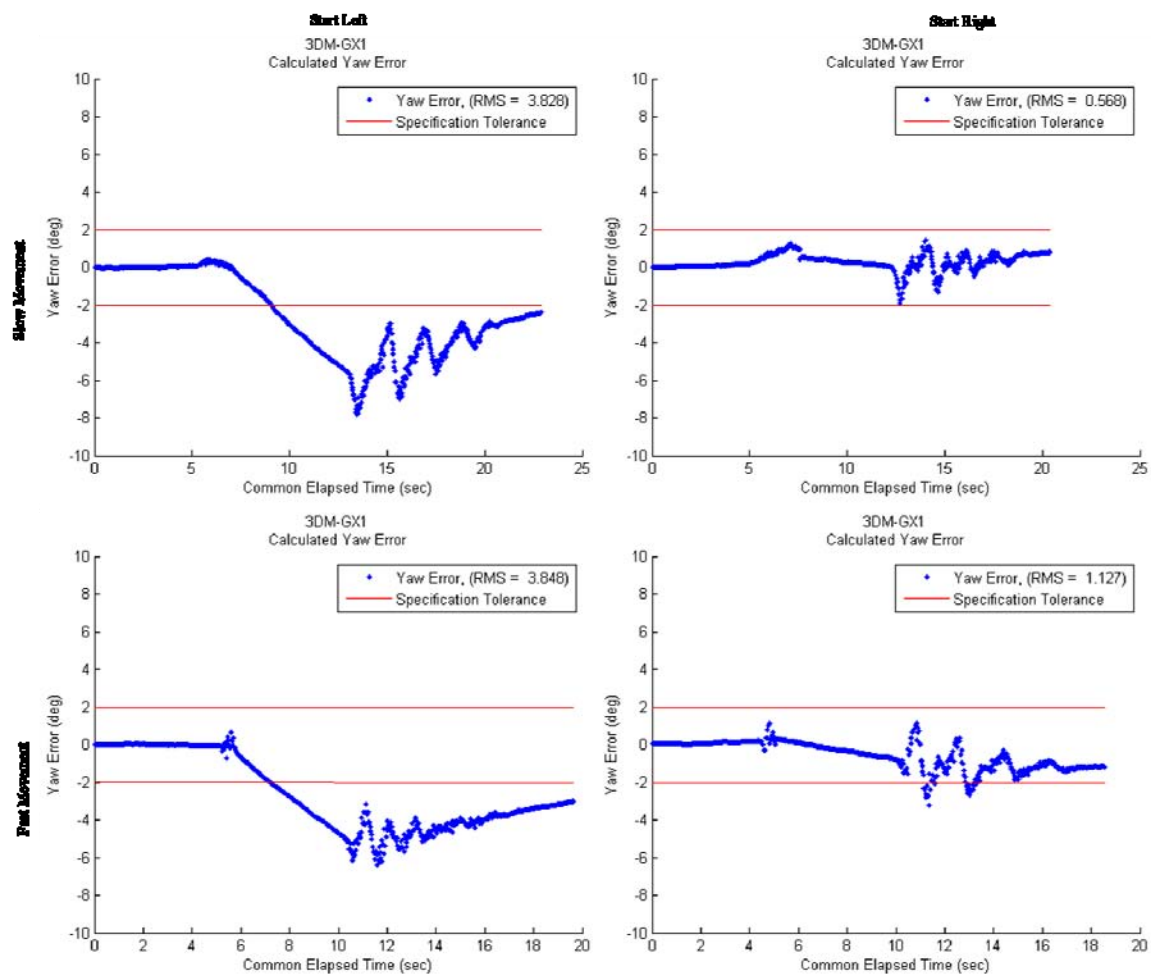


Figure 83. 3DM-GX1, Semi-Static Test, Yaw Accuracy.

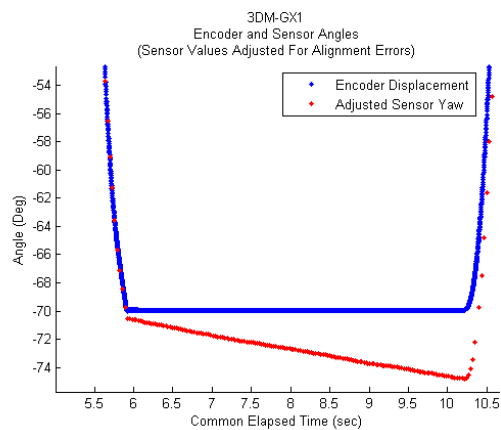


Figure 84. 3DM-GX1, Semi-Static Test, Yaw Accuracy, Zoomed in to Show Detail.

4. Free Swing With Impact, and Free Swing With Impact and Hold

The following sections contain the accuracy results of the Free Swing with Impact tests plotted next to the accuracy results of the corresponding Free Swing with Impact and Hold tests for comparison. All were performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the impact tests for the 3DM-GX1 are shown in Figures 85, 86, and 87 for the low, medium, and high release test cases, respectively. Both the impact and the impact with hold cases are shown in all plots.

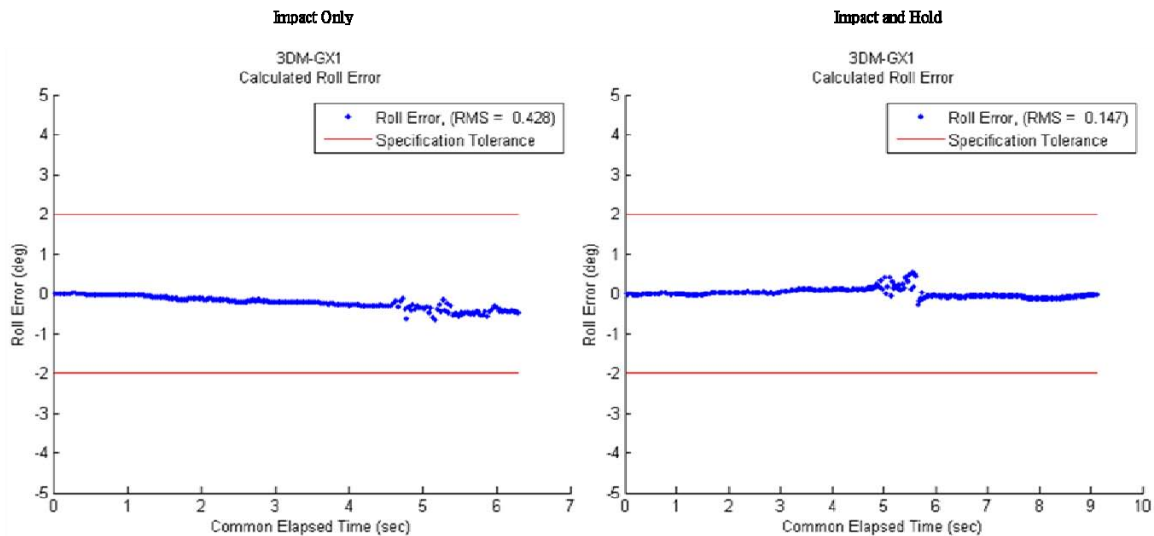


Figure 85. 3DM-GX1, Impact Tests, Roll Accuracy, Low Release Angle.

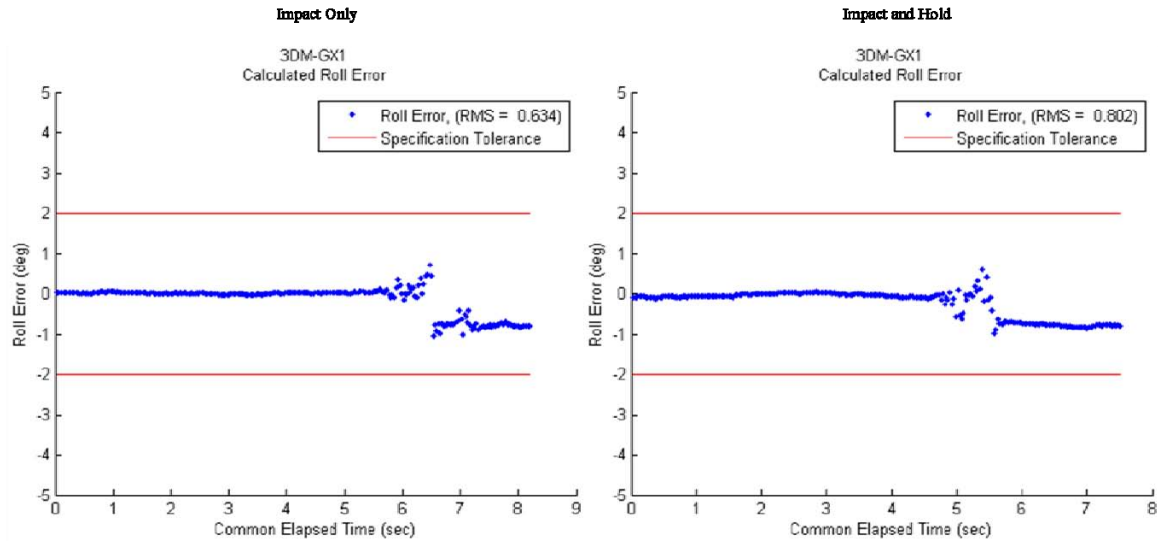


Figure 86. 3D-MGX1, Impact Tests, Roll Accuracy, Medium Release Angle.

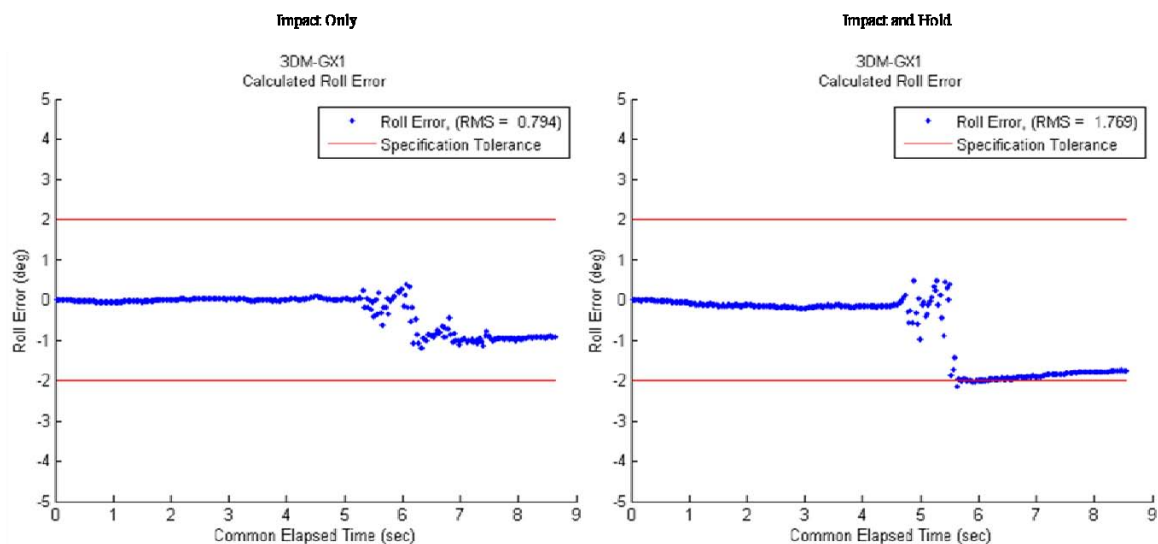


Figure 87. 3D-MGX1, Impact Tests, Roll Accuracy, High Release Angle.

b. Pitch

The pitch accuracy results of the impact tests for the 3D-MGX1 are shown in Figures 88, 89, and 90 for the low, medium, and high release test cases, respectively. Both the impact and the impact with hold cases are shown in all plots.

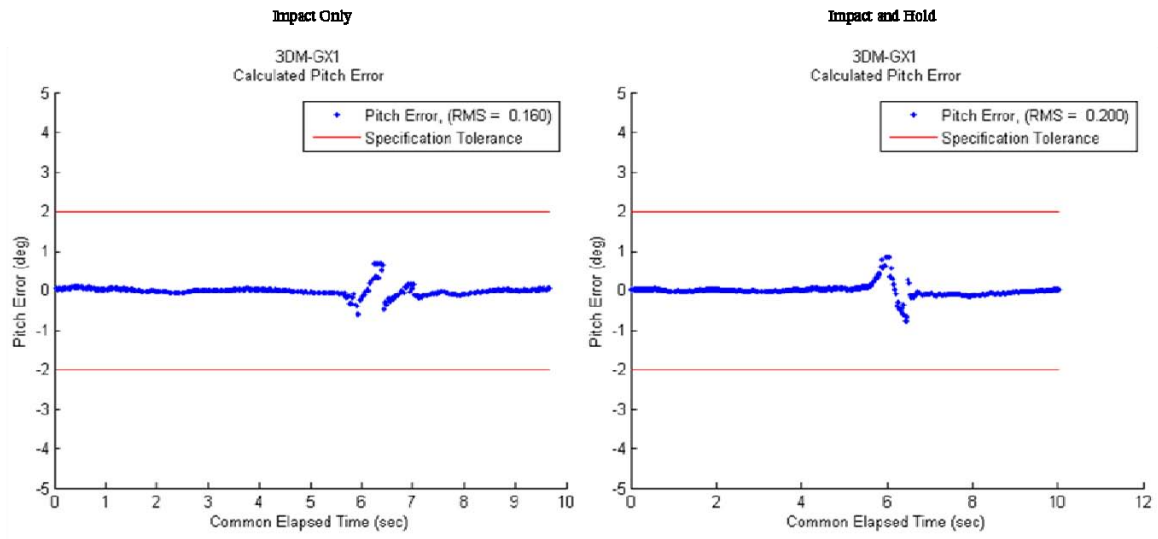


Figure 88. 3DM-GX1, Impact Tests, Pitch Accuracy, Low Release Angle.

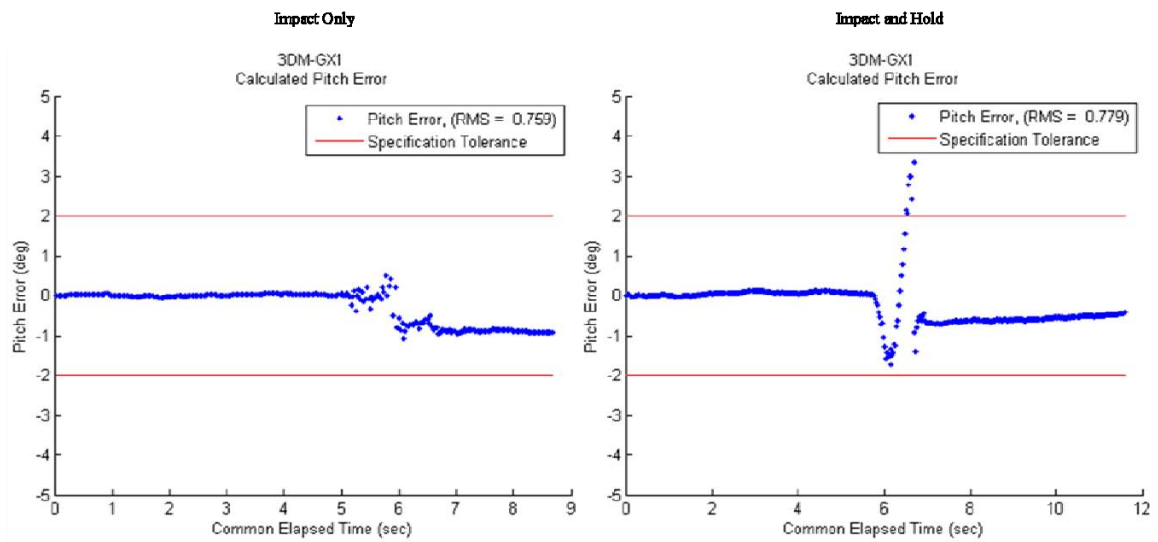


Figure 89. 3DM-GX1, Impact Tests, Pitch Accuracy, Medium Release Angle.

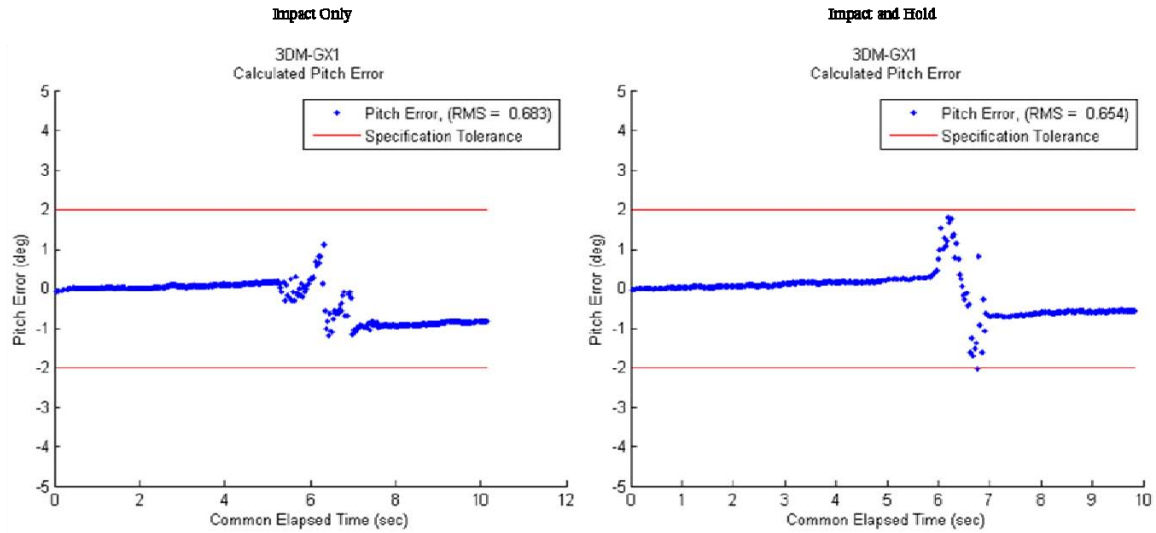


Figure 90. 3DM-GX1, Impact Tests, Pitch Accuracy, High Release Angle.

c. Yaw

The yaw accuracy results of the impact tests for the 3DM-GX1 are shown in Figures 91, 92, and 93 for the low, medium, and high release test cases, respectively. Both the impact and the impact with hold cases are shown in all plots.

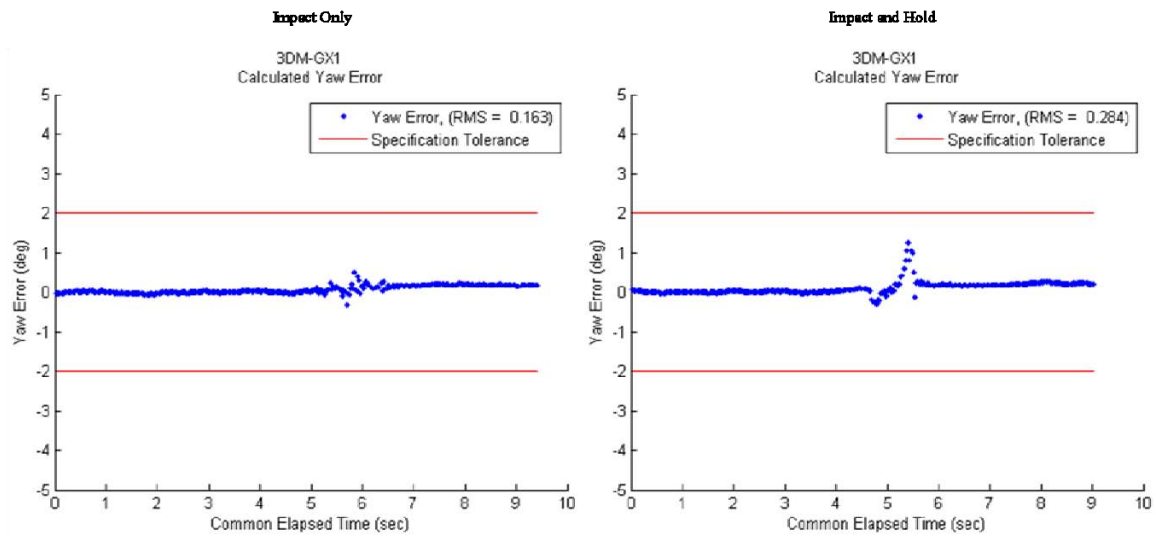


Figure 91. 3DM-GX1, Impact Tests, Yaw Accuracy, Low Release Angle.

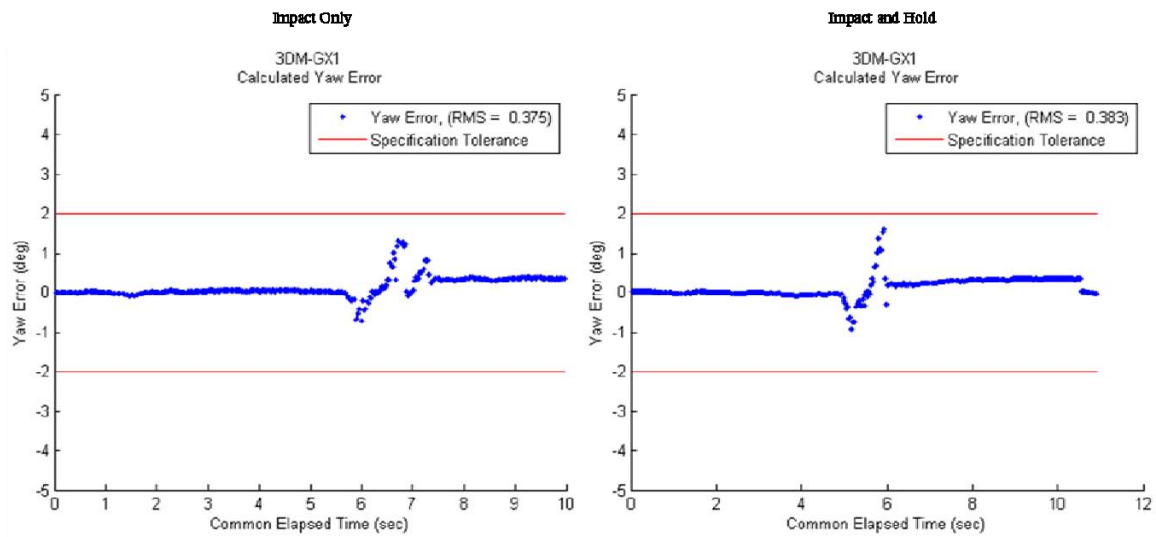


Figure 92. 3DM-GX1, Impact Tests, Yaw Accuracy, Medium Release Angle.

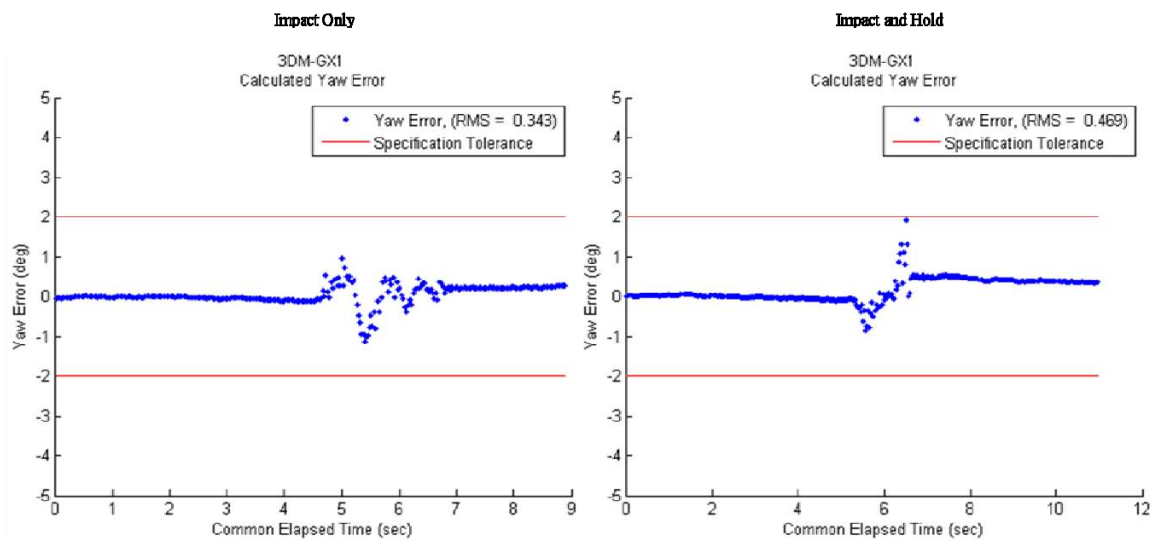


Figure 93. 3DM-GX1, Impact Tests, Yaw Accuracy, High Release Angle.

C. 3DM-GX3

This section contains the accuracy results of the tests performed according to the methodology set out in Chapter VI for the 3DM-GX3. For each test the roll, pitch, and yaw error plots are shown.

1. Free Swinging

The following sections contain the accuracy results of the Free Swinging tests performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the Free Swinging tests for the 3DM-GX3 are shown in Figures 94, 95, and 96 for the low, medium, and high release test cases, respectively.

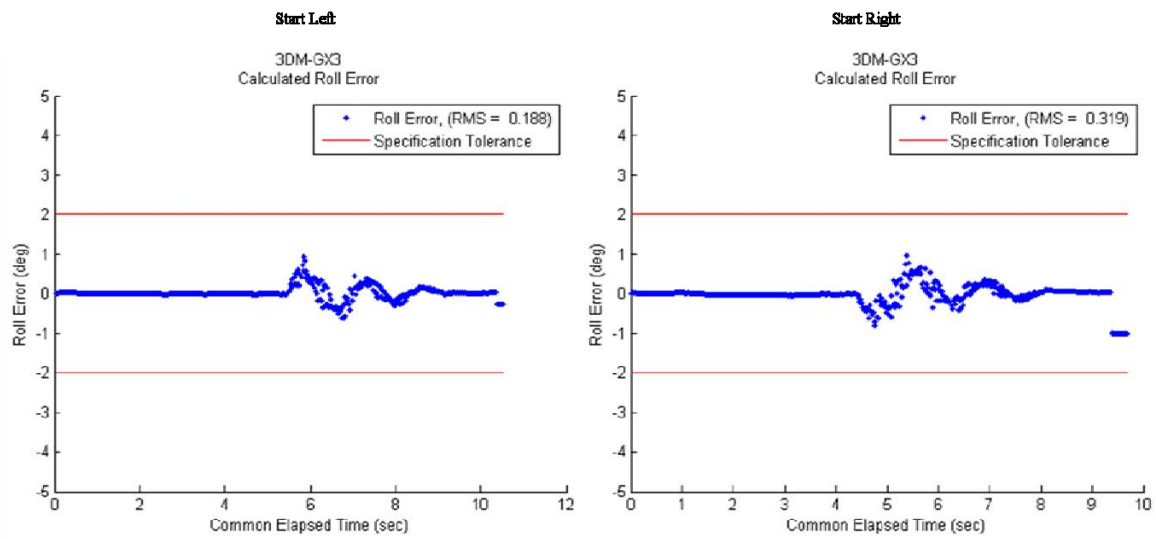


Figure 94. 3DM-GX3, Free Swinging Test, Roll Accuracy, Low Release Angle.

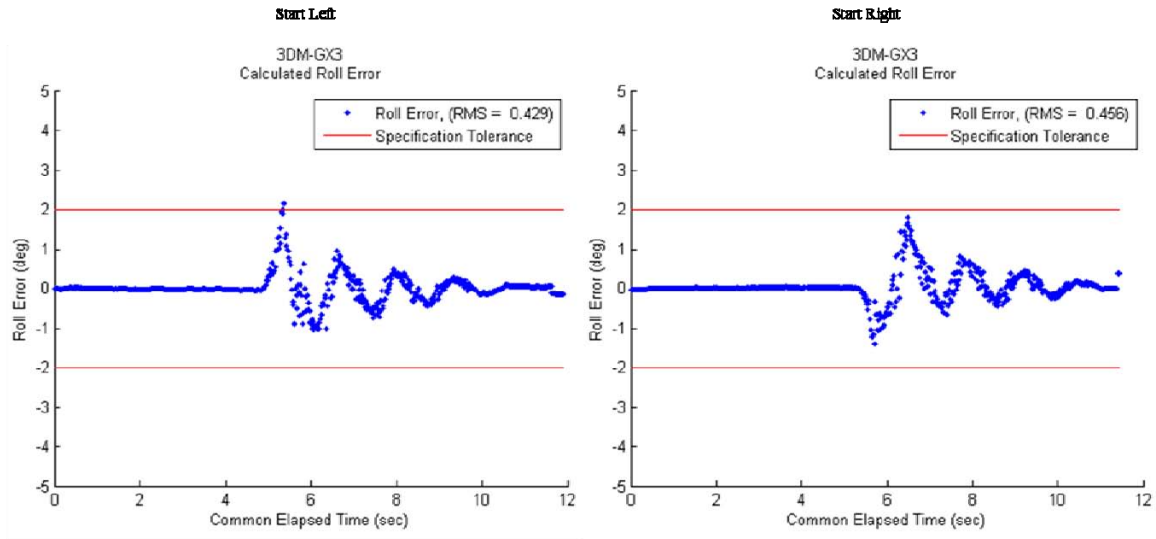


Figure 95. 3DM-GX3, Free Swinging Test, Roll Accuracy, Medium Release Angle.

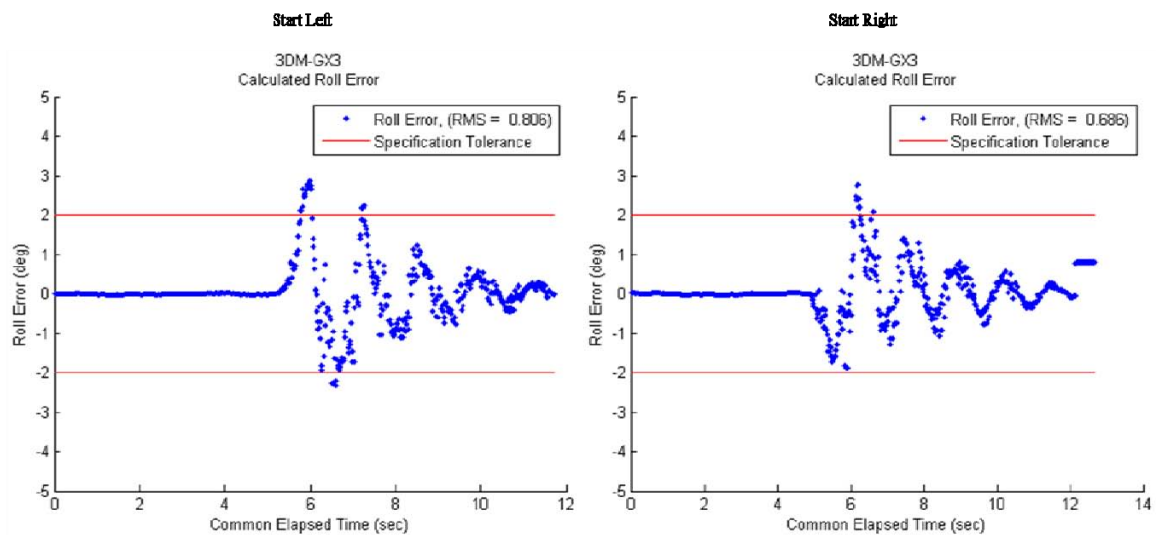


Figure 96. 3DM-GX3, Free Swinging Test, Roll Accuracy, High Release Angle.

b. Pitch

The pitch accuracy results of the Free Swinging tests for the 3DM-GX3 are shown in Figures 97, 98, and 99 for the low, medium, and high release test cases, respectively.

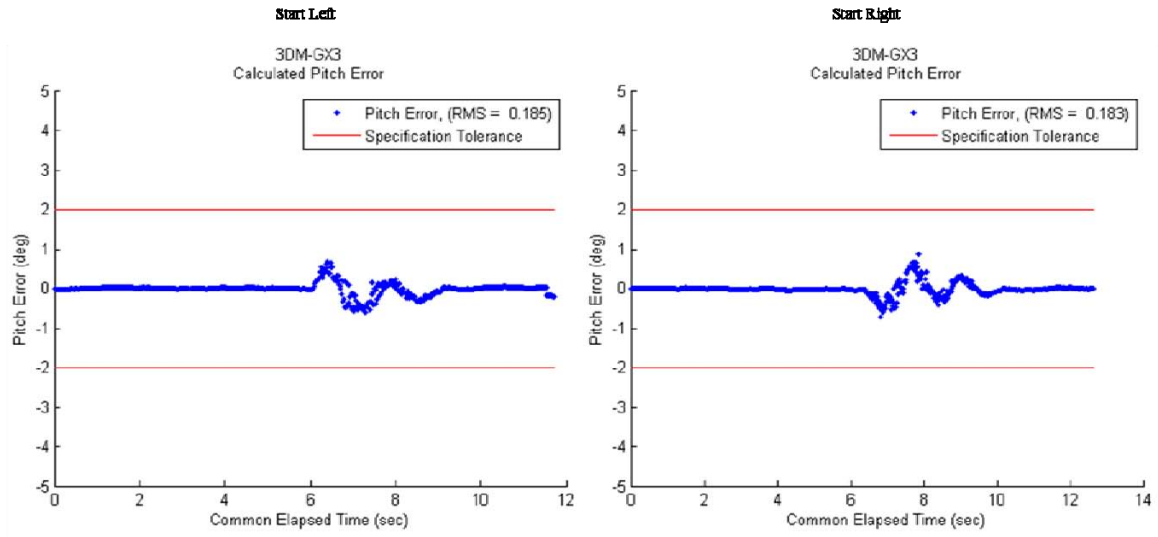


Figure 97. 3DM-GX3, Free Swinging Test, Pitch Accuracy, Low Release Angle.

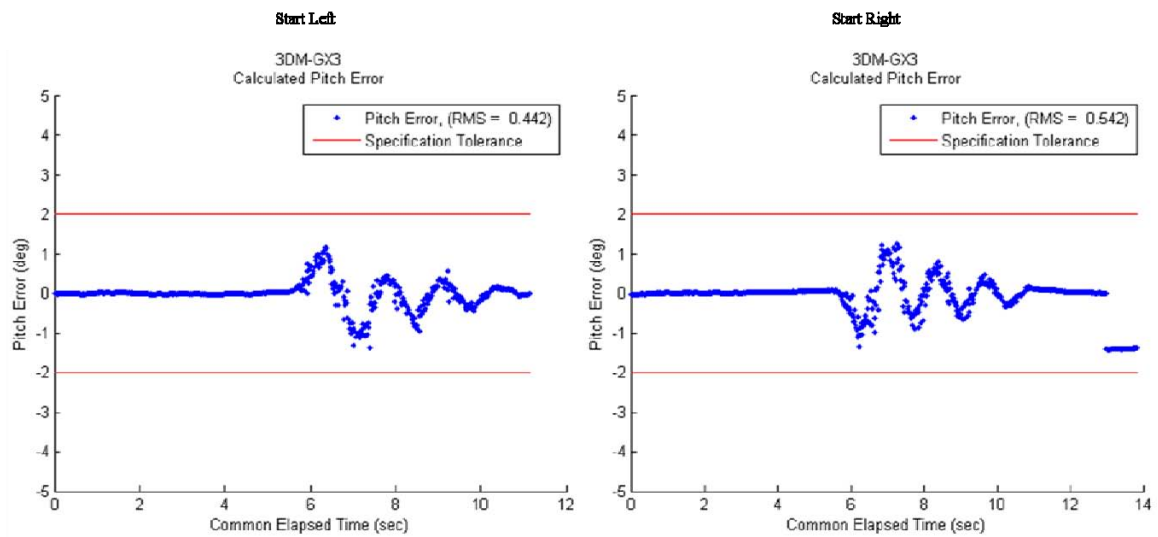


Figure 98. 3DM-GX3, Free Swinging Test, Pitch Accuracy, Medium Release Angle.

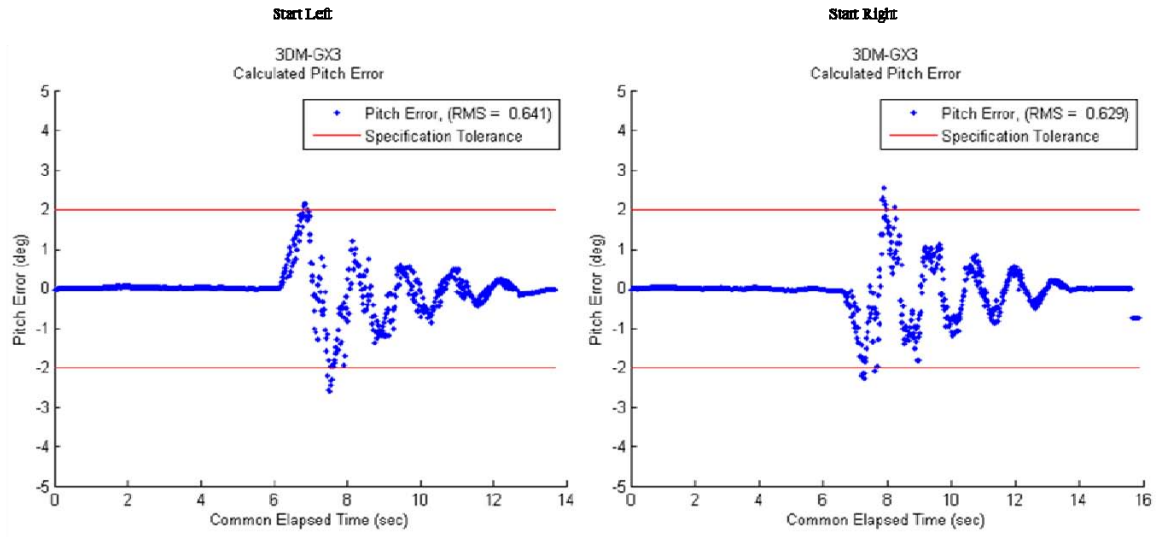


Figure 99. 3DM-GX3, Free Swinging Test, Pitch Accuracy, High Release Angle.

c. Yaw

The yaw accuracy results of the Free Swinging tests for the 3DM-GX3 are shown in Figures 100, 101, and 102 for the low, medium, and high release test cases, respectively.

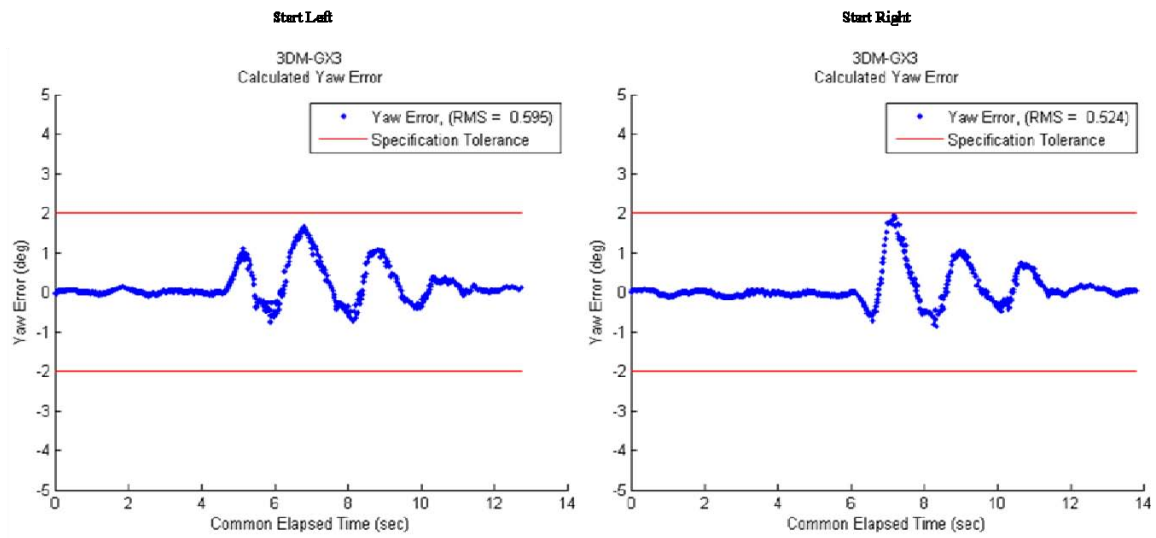


Figure 100. 3DM-GX3, Free Swinging Test, Yaw Accuracy, Low Release Angle.

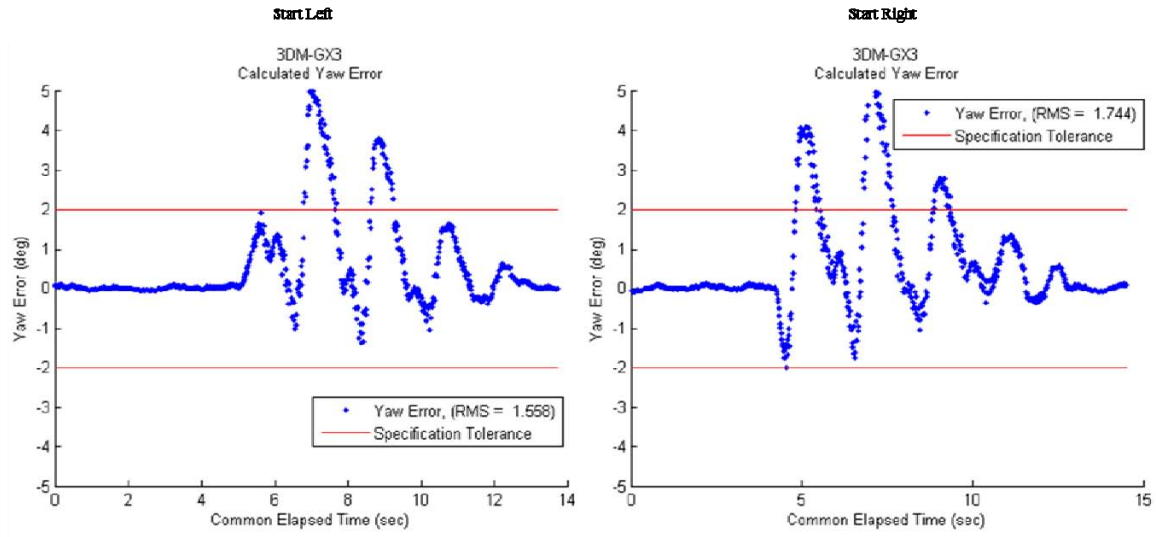


Figure 101. 3DM-GX3, Free Swinging Test, Yaw Accuracy, Medium Release Angle.

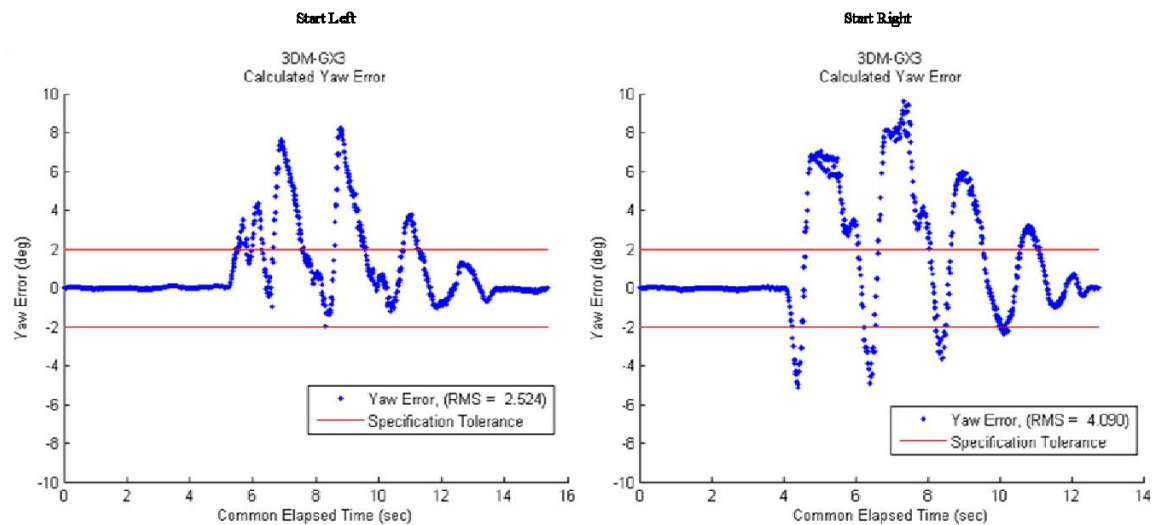


Figure 102. 3DM-GX3, Free Swinging Test, Yaw Accuracy, High Release Angle.

d. Free Swinging Tests' Cross-Talk

Although attempts were made to isolate each axis under test, some amount of cross-talk was seen. This was partly due to the test apparatus setup and partly due to errors in the sensor. In Figures 103, 104, and 105, the unadjusted roll, pitch, and yaw are

all plotted on the same figure, one figure for each test apparatus configuration. The selected values for all three plots are shown for a “high” release case.

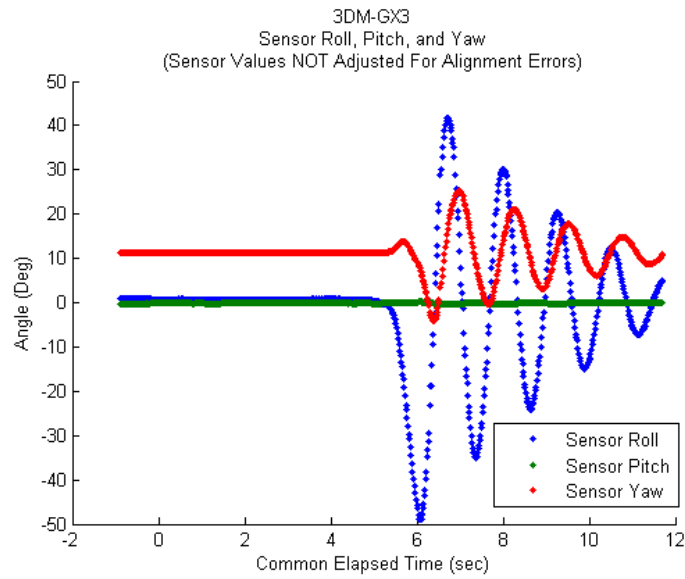


Figure 103. 3DM-GX3, Free Swinging Test, Cross-Talk Plot, Roll Under Test.

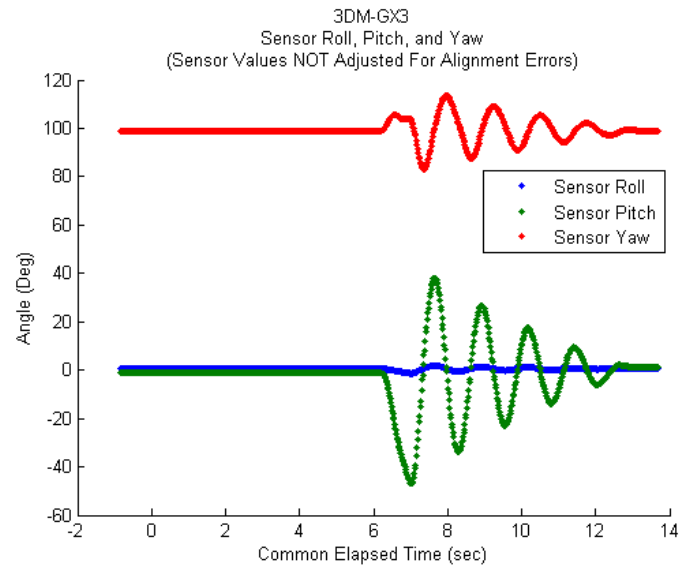


Figure 104. 3DM-GX3, Free Swinging Test, Cross-Talk Plot, Pitch Under Test.

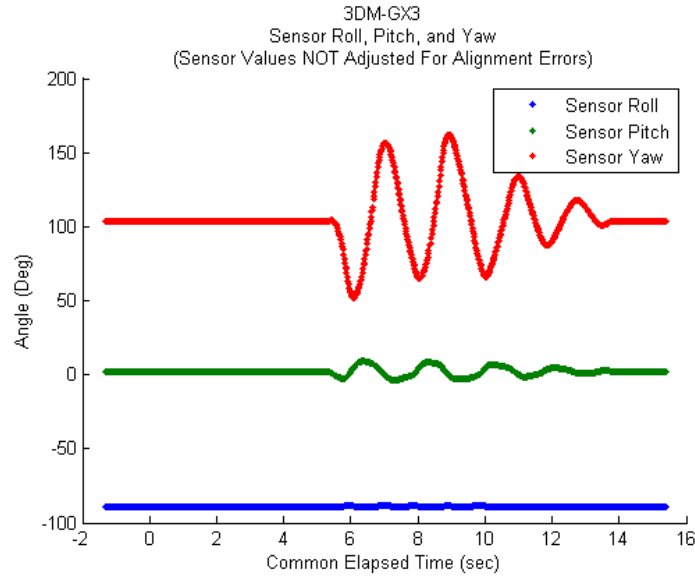


Figure 105. 3D-MGX3, Free Swinging Test, Cross-Talk Plot, Yaw Under Test.

2. Arbitrary swinging

The following sections contain the accuracy results of the Arbitrary Swinging tests performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the Arbitrary Swinging tests for the 3D-MGX3 are shown in Figure 106. Both the slow and the fast movement accuracies are shown.

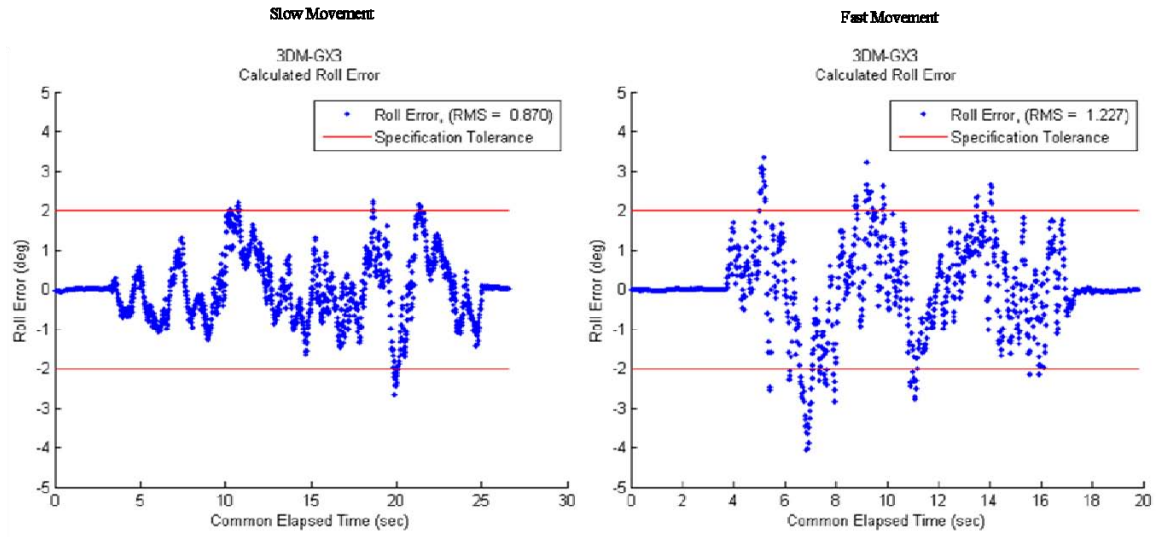


Figure 106. 3DM-GX3, Arbitrary Swinging Test, Roll Accuracy.

b. Pitch

The pitch accuracy results of the Arbitrary Swinging tests for the 3DM-GX3 are shown in Figure 107. Both the slow and the fast movement accuracies are shown.

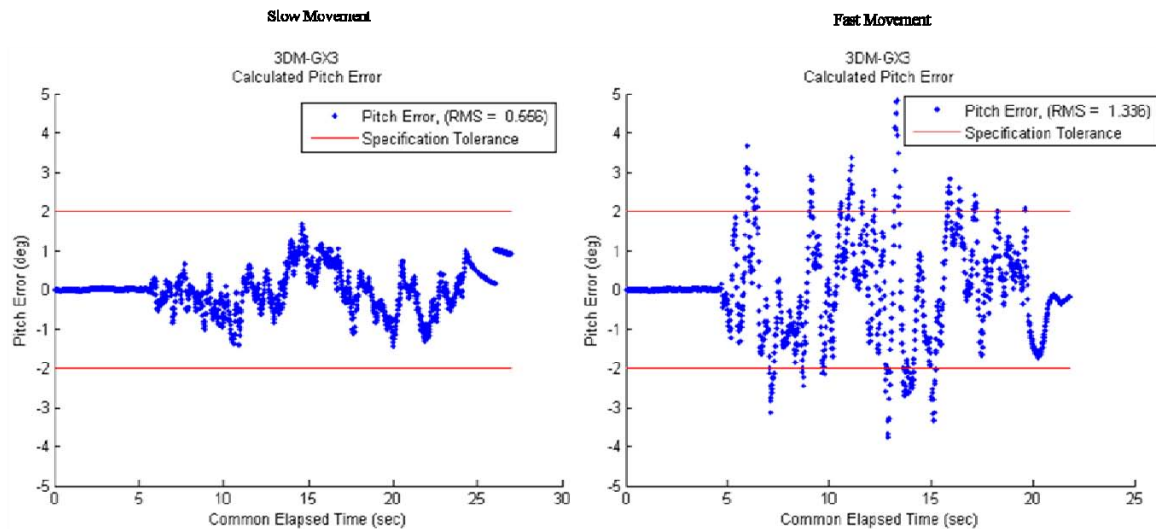


Figure 107. 3DM-GX3, Arbitrary Swinging Test, Pitch Accuracy.

c. Yaw

The yaw accuracy results of the Arbitrary Swinging tests for the 3DM-GX1 are shown in Figures 108. Both the slow and the fast movement accuracies are shown.

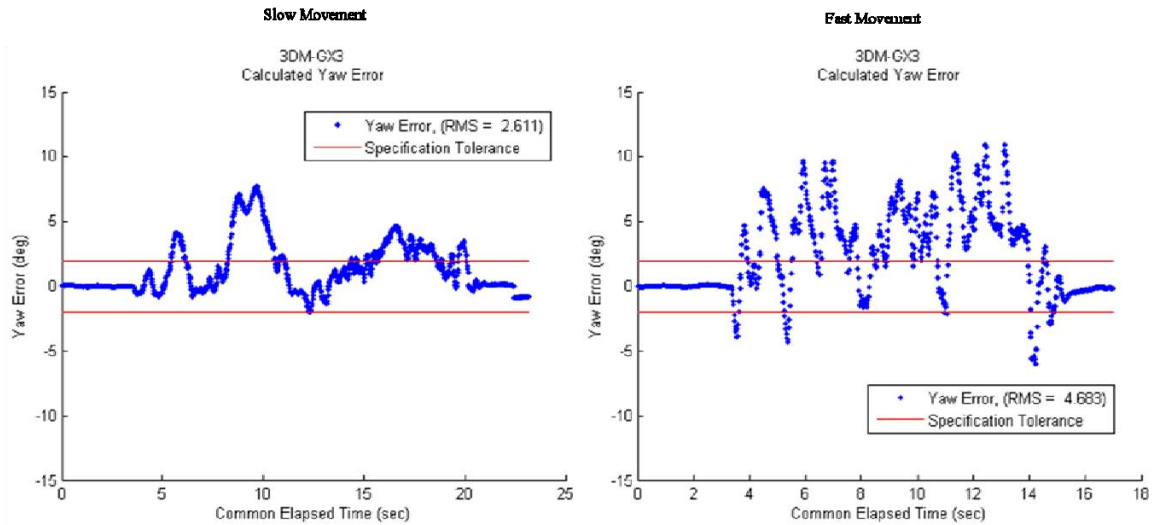


Figure 108. 3DM-GX3, Arbitrary Swinging Test, Yaw Accuracy.

3. Semi-Static: Move and Hold

The following sections contain the accuracy results of the Semi-Static tests performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the Semi-Static tests for the 3DM-GX3 are shown in Figure 109 for both the slow and fast cases starting to the left and to the right. In Figure 110, the point where maximum deflection is reached when moving quickly to the right is zoomed in to show in detail the behavior of the sensor at that point.

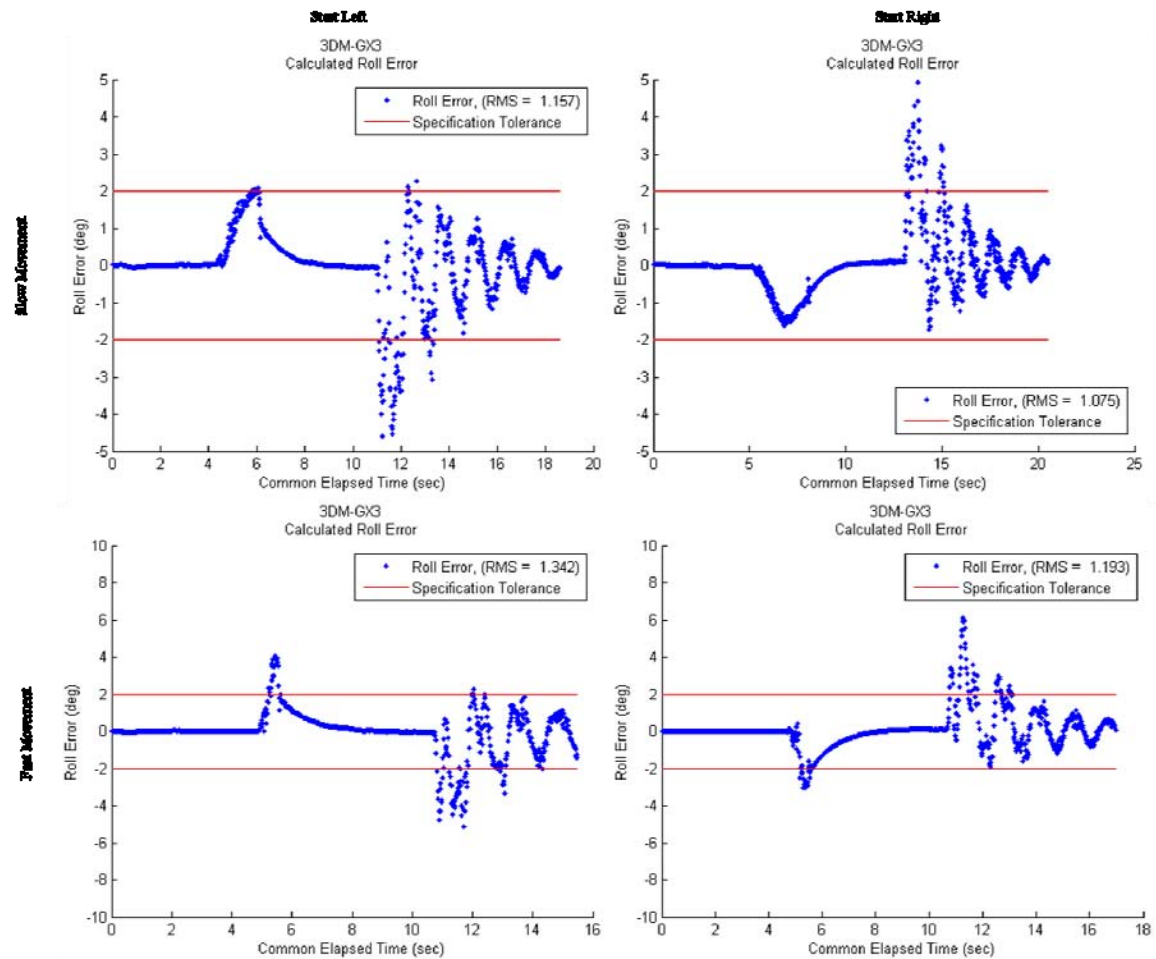


Figure 109. 3DM-GX3, Semi-Static Test, Roll Accuracy.

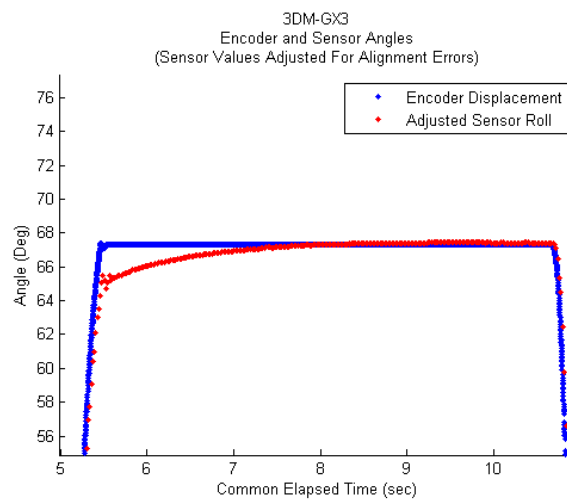


Figure 110. 3DM-GX3, Semi-Static Test, Roll Accuracy, Zoomed in to Show Detail.

b. Pitch

The pitch accuracy results of the Semi-Static tests for the 3DM-GX3 are shown in Figure 111 for both the slow and fast cases starting to the left and to the right. In Figure 112, the point where maximum deflection is reached when moving quickly to the right is zoomed in to show in detail the behavior of the sensor at that point.

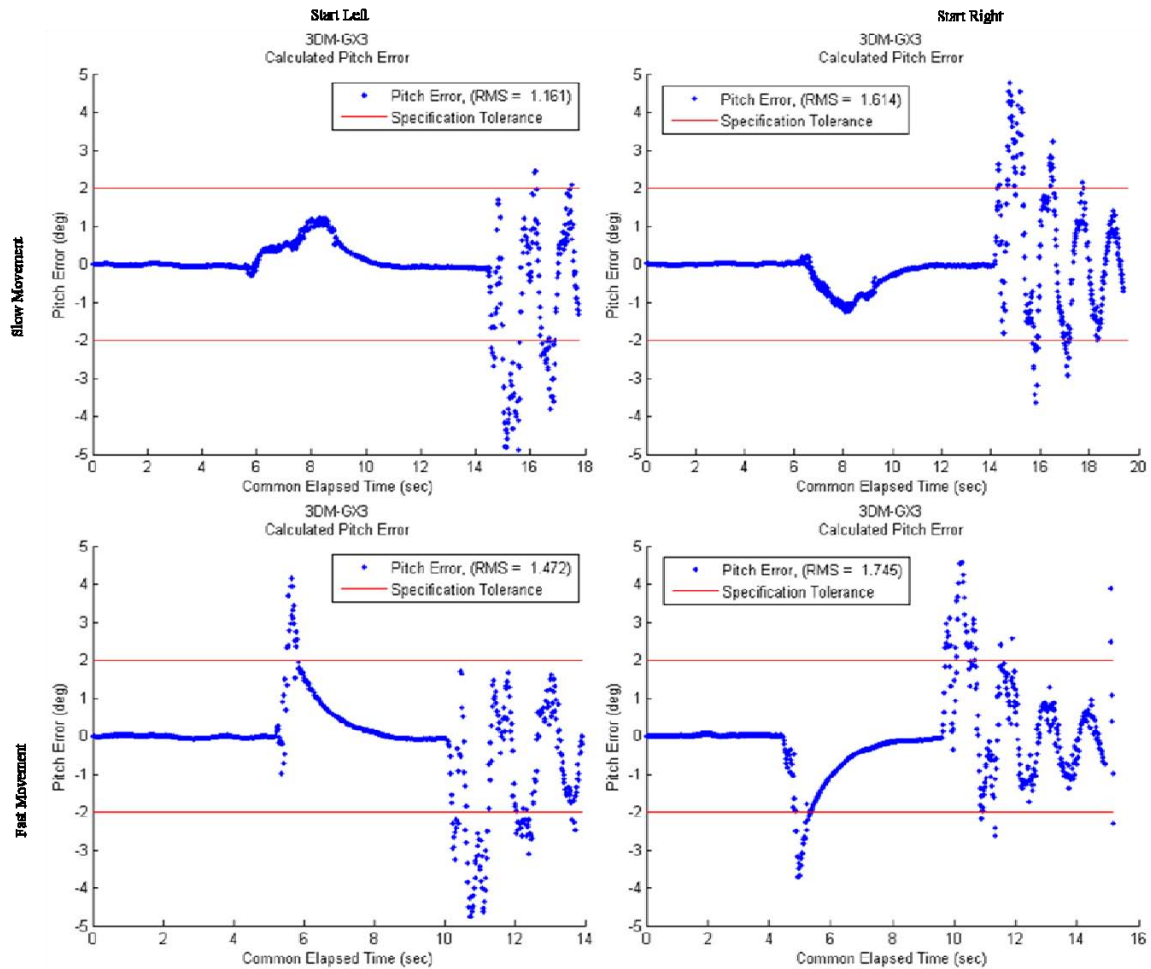


Figure 111. 3DM-GX3, Semi-Static Test, Pitch Accuracy.

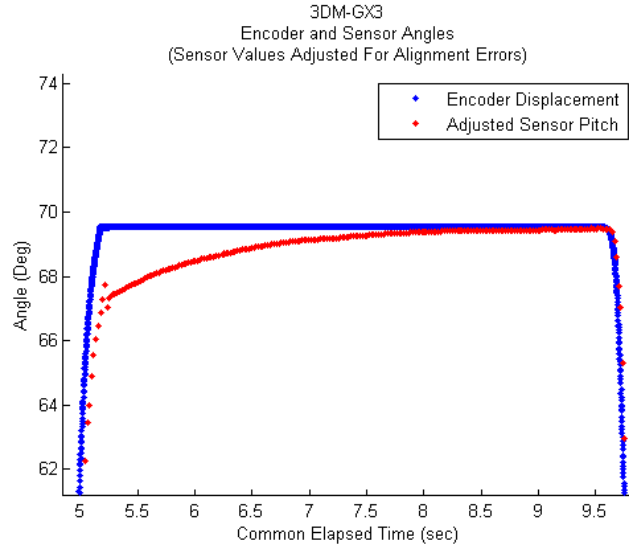


Figure 112. 3DM-GX3, Semi-Static Test, Pitch Accuracy, Zoomed in to Show Detail.

c. Yaw

The yaw accuracy results of the Semi-Static tests for the 3DM-GX3 are shown in Figure 113 for both the slow and fast cases starting to the left and to the right. In Figure 114, the point where maximum deflection is reached when moving quickly to the right is zoomed in to show in detail the behavior of the sensor at that point.

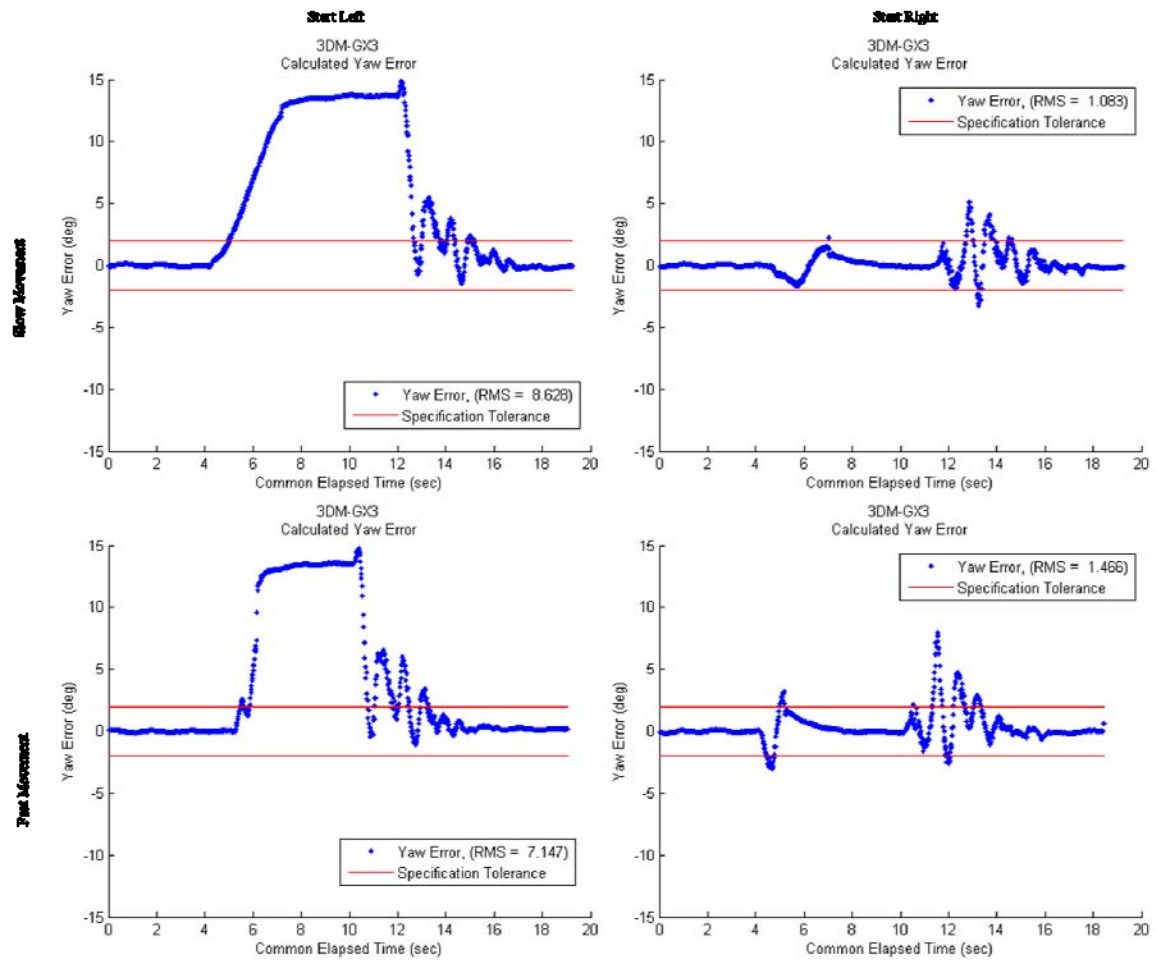


Figure 113. 3DM-GX3, Semi-Static Test, Yaw Accuracy.

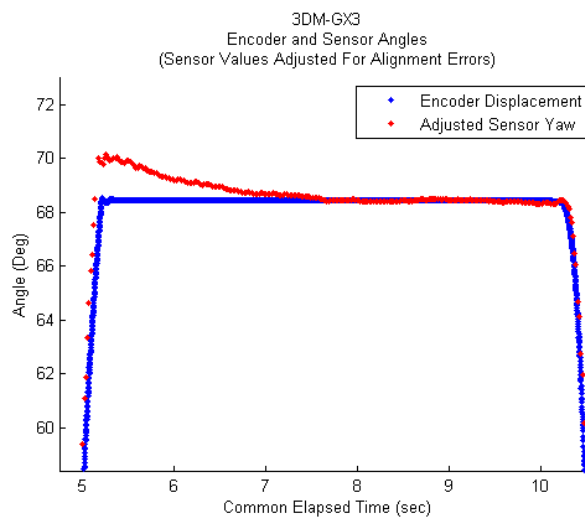


Figure 114. 3DM-GX3, Semi-Static Test, Yaw Accuracy, Zoomed in to Show Detail.

4. Free Swing With Impact, and Free Swing With Impact and Hold

The following sections contain the accuracy results of the Free Swing with Impact tests plotted next to the accuracy results of the corresponding Free Swing with Impact and Hold tests for comparison. All were performed according to the methodology set out in Chapter VI.

a. Roll

The roll accuracy results of the impact tests for the 3DM-GX3 are shown in Figures 115, 116, and 117 for the low, medium, and high release test cases, respectively. Both the impact and the impact with hold cases are shown in all plots.

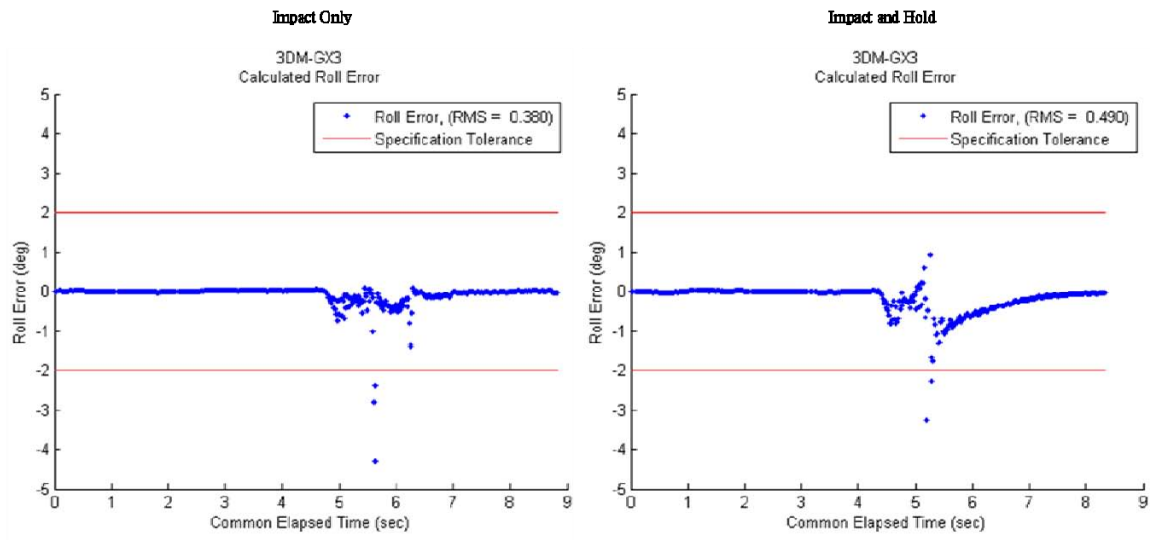


Figure 115. 3DM-GX1, Impact Tests, Roll Accuracy, Low Release Angle.

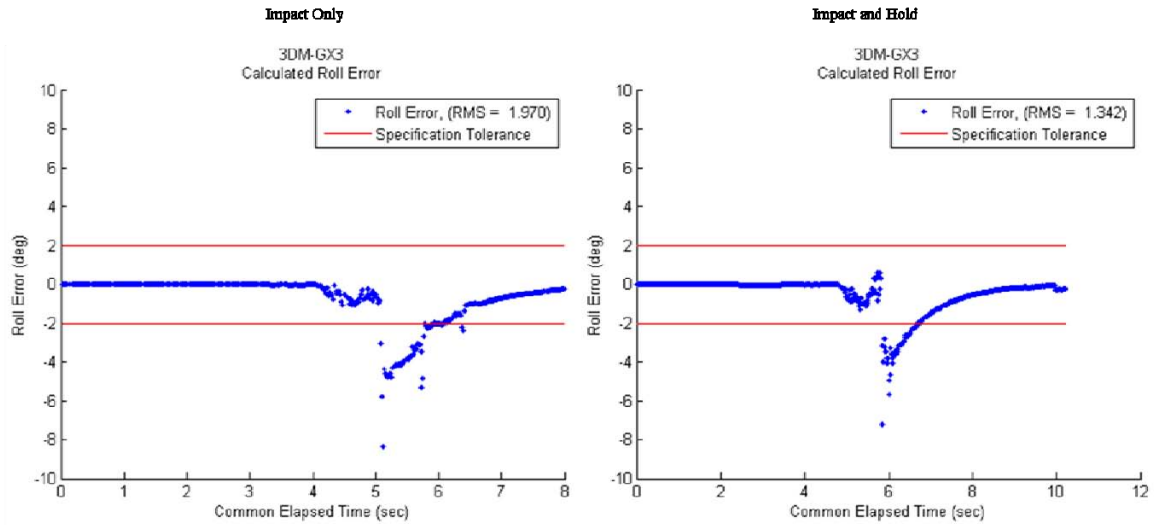


Figure 116. 3DM-GX1, Impact Tests, Roll Accuracy, Medium Release Angle.

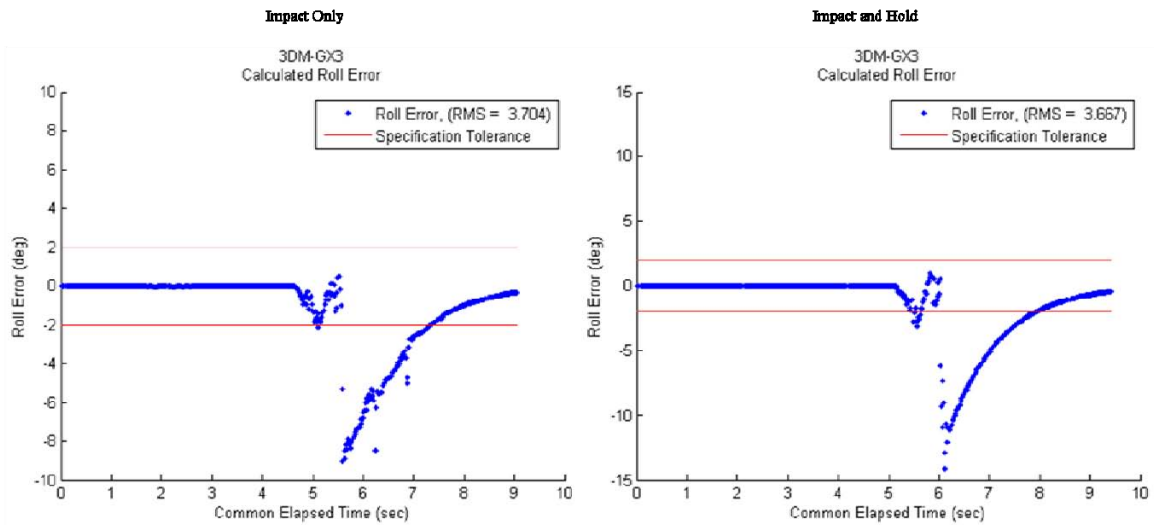


Figure 117. 3DM-GX1, Impact Tests, Roll Accuracy, High Release Angle.

b. Pitch

The pitch accuracy results of the impact tests for the 3DM-GX3 are shown in Figures 118, 119, and 120 for the low, medium, and high release test cases, respectively. Both the impact and the impact with hold cases are shown in all plots.

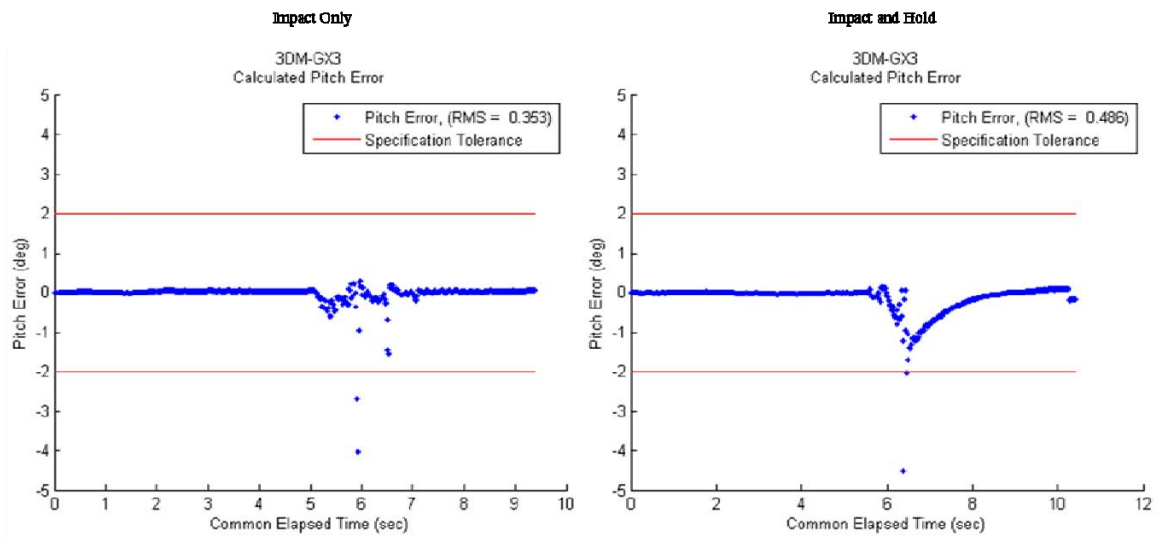


Figure 118. 3DM-GX1, Impact Tests, Pitch Accuracy, Low Release Angle.

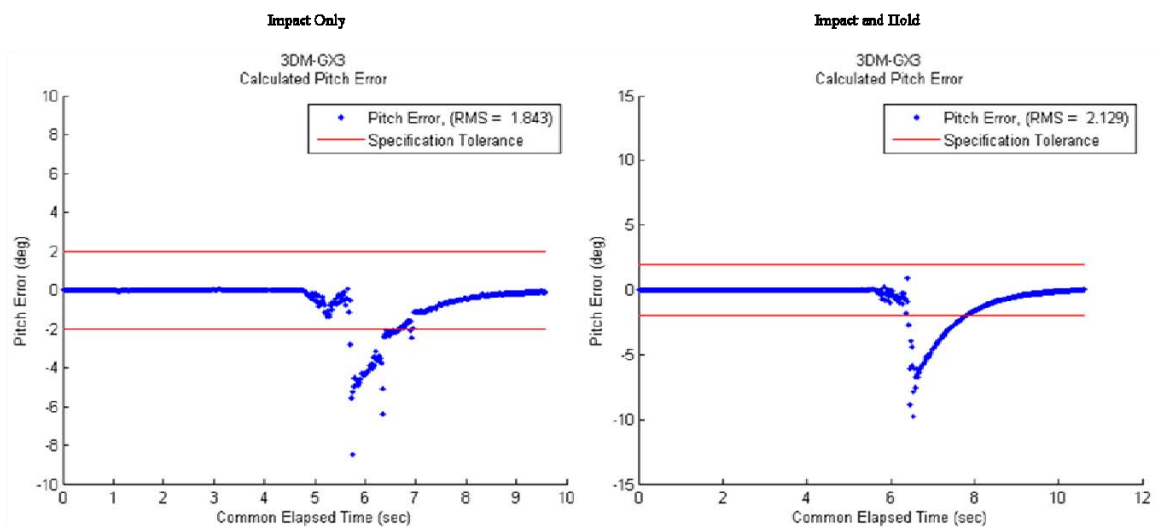


Figure 119. 3DM-GX1, Impact Tests, Pitch Accuracy, Medium Release Angle.

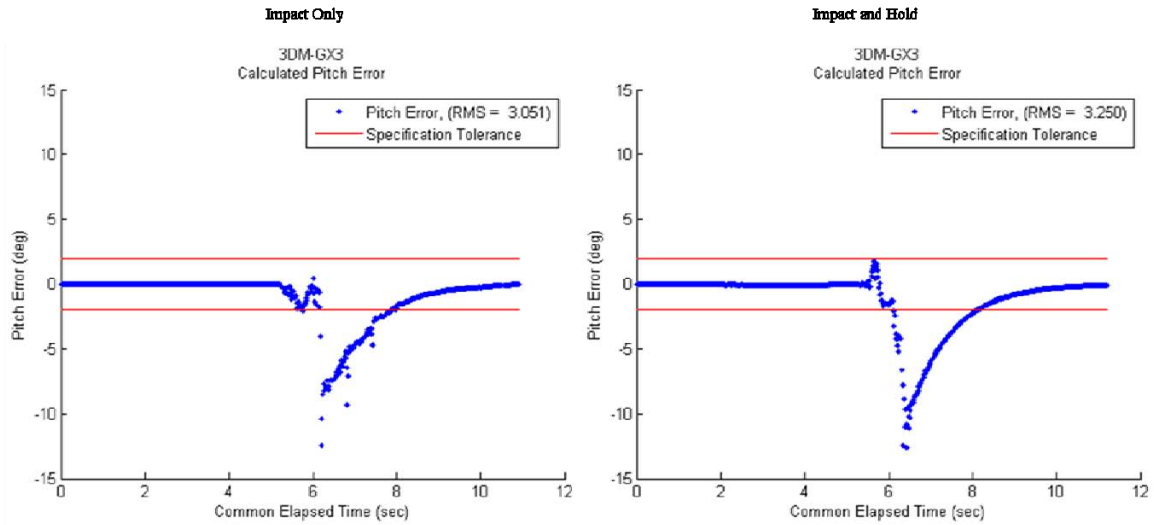


Figure 120. 3DM-GX1, Impact Tests, Pitch Accuracy, High Release Angle.

c. Yaw

The yaw accuracy results of the impact tests for the 3DM-GX1 are shown in Figures 121, 122, and 123 for the low, medium, and high release test cases, respectively. Both the impact and the impact with hold cases are shown in all plots.

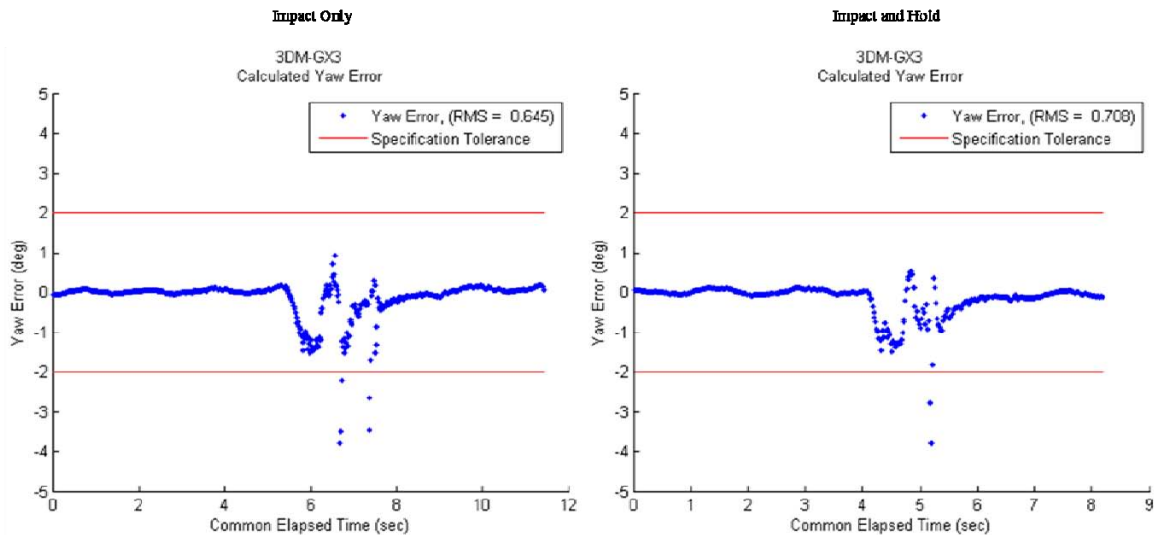


Figure 121. 3DM-GX3, Impact Tests, Yaw Accuracy, Low Release Angle.

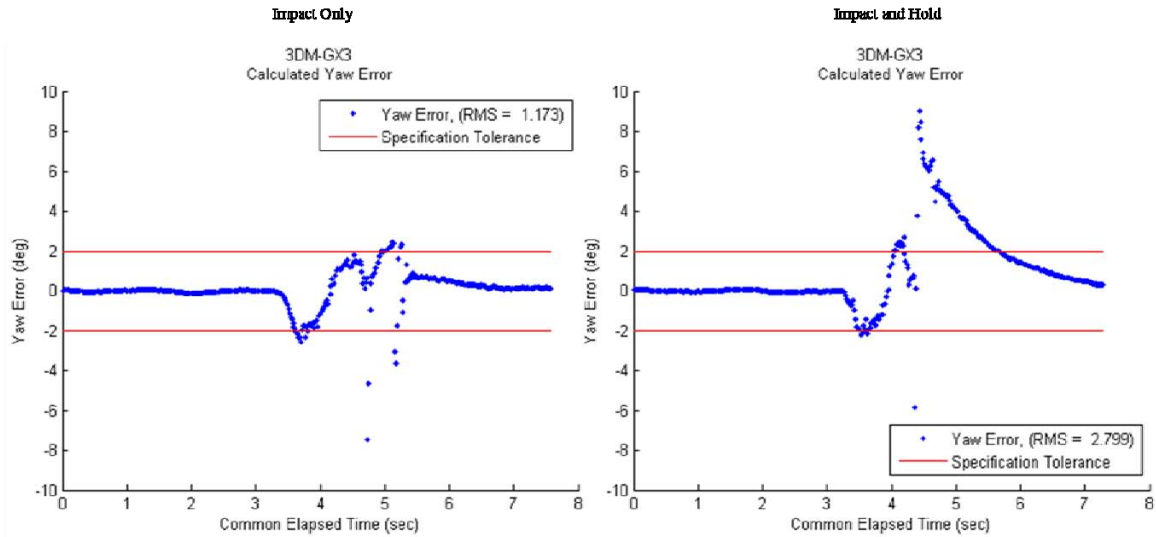


Figure 122. 3DM-GX3, Impact Tests, Yaw Accuracy, Medium Release Angle.

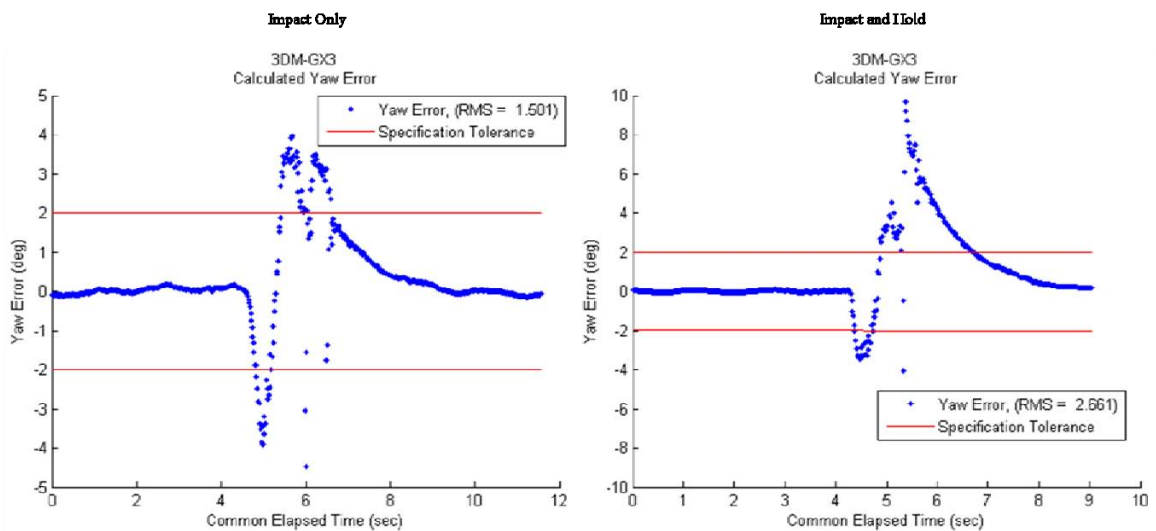


Figure 123. 3DM-GX3, Impact Tests, Yaw Accuracy, High Release Angle.

D. OBSERVATIONS

As the data show, both the 3DM-GX1 and 3DM-GX3 performed fairly well in the dynamic tests and often met the RMS dynamic accuracy requirements. However, there were some cases where the sensors either did not meet the specifications or had large spikes in the error. From the data presented, certain trends were identified for both of the sensors individually and as a pair.

The 3DM-GX1 usually performed well through both slow and fast motion tests but experienced significant drift when stopped following dynamic motion. This behavior was most pronounced in the semi-static test cases. In such test cases, the error tended to grow over time, even though the sensor was not moving. In addition, the impact tests seemed to cause discontinuities and jumps in the error of up to a full degree following impact, and the accuracy drifted from there. Otherwise, the 3DM-GX1 seemed to track well on the axis under test through dynamic motions, and there did not appear to be much cross-talk from either roll or pitch to yaw. Cross-talk of less than five degrees was observed that was more likely due to apparatus alignment errors than a serious sensor issue.

The 3DM-GX3 did not perform as well as the 3DM-GX1 in most tests, usually having higher overall RMS error and larger swings in the errors. The 3DM-GX3 seemed to have a filter that delayed the sensor's responsiveness to quick movements since the errors were low with slower motions but tended to grow with increasing pendulum velocity. Also, the values output by the sensor tended to undershoot the peak truth values, especially with fast dynamic motions, as though the data did not quite catch up before the sensor was moving down again. This delayed behavior was most evident on the semi-static tests, where the data took a few seconds to reach the true value. Also, there appeared to be significant cross-talk between the roll or pitch and yaw axes. When testing either roll or pitch, the heading of the test apparatus was not changed, yet the yaw output registered 10 to 15 degree swings. The magnitude of the yaw values indicated that the cause was the sensor's hardware or algorithms and not simply a misalignment of the test apparatus.

Some of the errors seen in the data were common to both sensors. For both the 3DM-GX1 and the 3DM-GX3, the yaw accuracy tended to be the worst for each type of test that was run compared to the roll or the pitch accuracy. Also, both sensors experienced discontinuities in the error with the impact testing. The 3DM-GX1 had less error but was more likely to maintain an offset after impact, whereas the initial errors for the 3DM-GX3 were much larger on impact, upwards of five to ten degrees, but tended to decay to zero within a few seconds.

Overall, the 3DM-GX1 was more likely to be within the stated ± 2 degree expected accuracy than the 3DM-GX3, both on average and throughout the dynamic motions. However, as was shown, certain dynamic motions that may be encountered in personal navigation or human motion tracking caused both sensors' errors to exceed the manufacturer's listed accuracy specification. Though a small portion of the errors may be due to the test setup, it is believed that the plots shown are a fair representation of the expected accuracies of these sensors through these motions. Since the primary goal of this thesis was to develop the test apparatus and methodology and to show that the tests worked, not to fully test the MicroStrain sensors for specification compliance, the manufacturer was not contacted for information regarding these results.

E. SUMMARY

In this chapter, the accuracy analysis plots were presented of the data collected using the new test apparatus with the data collection setup described in previous chapters. The data were presented such that conclusions regarding the trends of each sensor could be analyzed and understood, and so that a simple side-by-side comparison of the two sensors could be made.

The thesis is concluded in the next chapter by summing up what was accomplished. Based on the data presented in this chapter, recommendations are made regarding the potential use of the 3DM-GX1 and 3DM-GX3 in either personal navigation or human motion tracking. Finally, recommendations are made for future work based on topics and capabilities introduced in this thesis.

VIII. CONCLUSIONS

The thesis is concluded in this chapter by reviewing what was accomplished and showing that the initial goals of the thesis were all met. In addition, some conclusions are drawn regarding the “best” sensor for personal navigation and for immersive virtual reality training based on the data shown in Chapter VII, the 3DM-GX1 or the 3DM-GX3. Finally, topics of future work to expand on the findings of this thesis are presented.

A. WHAT WAS ACCOMPLISHED

In this thesis, a method to test the dynamic accuracy of MEMS MARG sensors for use in personal navigation or human motion tracking for immersive virtual reality training was successfully developed and implemented. A sturdy, low-cost test apparatus was designed and built, and a series of tests were developed to mimic certain aspects of the dynamic motions expected to be encountered by the sensors in practical applications. The test apparatus used an optical encoder to precisely gather “truth” data regarding the orientation of the sensor under test, and therefore, accurate and repeatable test data were able to be gathered. A data collection system using a National Instruments CompactRIO with an FPGA and implementing LabVIEW virtual instruments worked well enough to collect and synchronize the necessary data. Post processing and analysis of the data was made simple through the creation of a MATLAB GUI. Finally, a series of tests were run to confirm the functionality of the test apparatus as well as to characterize the accuracy of the MicroStrain 3DM-GX1 and 3DM-GX3 sensors and to identify if either sensor was better suited for a particular dynamic application than the other.

B. THE “BEST” SENSOR

The data plots in Chapter VII showed that although both sensors met the RMS dynamic accuracy specifications most of the time, there were certain motions and certain orientations that caused the accuracy to go outside of the bounds. In addition, certain dynamic motions caused the error to spike outside of the ± 2 degree tolerance even though the overall RMS accuracy was within tolerances.

1. 3DM-GX1 Anomalies and Impact

The 3DM-GX1 tended to do very well tracking the orientation of the sensor through all of the dynamic motion tests, and of the three orientations tested the yaw orientation accuracy was the often the worst. Overall, the 3DM-GX1 consistently outperformed the 3DM-GX3 in free swinging, arbitrary swinging, and impact tests. Its biggest problem was the tendency to drift following dynamic motions, seen clearly in the semi-static tests. Additionally, medium and high impact tests introduced discontinuous jumps in the error of up to one degree, and this error was maintained through the remainder of the test. Whereas the 3DM-GX3 accuracy errors tended to go to zero over time, the 3DM-GX1 error grew over time following dynamic motion. For short-term windows of data, the GX1 appeared to be accurate in measuring the change in orientation.

The results imply that the 3DM-GX1 would be good at measuring orientation for short spurts, but that the drift could cause significant problems in personal navigation or motion tracking applications in the long term. However, if the drift were somehow removed or compensated for, then this would be an adequate sensor for both of these major applications. In addition, depending upon how the yaw calculations were used, a significant heading error could also accumulate over time. Finally, the discontinuous errors could cause significant problems if used to measure something with significant impact, such as a foot sensor for personal navigation.

2. 3DM-GX3 Anomalies and Impact

Although the RMS accuracy results were typically within the ± 2 degree bounds, the 3DM-GX3 appeared to perform worse than the 3DM-GX1 when looking at variation in the dynamic accuracy of the sensor's orientation. There was much more variation than the 3DM-GX1, and often the data measured did not go all of the way to the peak values measured by the encoder, creating a sinusoidal error that was not related to the timing. It appeared that the GX3 was filtering data and that the output could not keep up with the dynamic motions. For slow dynamic tests, the sensor performed fairly well. However, the error consistently grew as the dynamics increased. This was most evident in the semi-

static tests. When the pendulum arm was drawn to one side, it took a couple of seconds for the errors to decay to zero. This was also seen on the impact tests. On impact a large (>10 degree) discontinuous error showed up, which gradually died out to zero.

In addition to the problems keeping up with fast movements, the 3DM-GX3 had significant cross-talk between the different orientations. When measuring pitch or roll, the reported yaw changed by 10–15 degrees even though the heading of the sensor remained constant. The yaw also experienced an anomaly on the semi-static test to the left (counter clockwise) in that a nearly 15 degree error occurred at maximum deflection, though no similar error was seen when moving the other direction. The anomaly was not investigated further due to time constraints.

Since the data pulled from the 3DM-GX3 in the tested configuration appeared to have a long delay when responding to dynamic motions, the configured sensor would not be well suited to applications where the orientation was changing rapidly. Since the error tended to go to zero over time, it did offer some advantages over the 3DM-GX1 in slow moving or semi-static applications. In addition, the problem of cross-talk between yaw and either roll or pitch could cause significant heading errors through a normal range of dynamic motions in a personal navigation or an immersive virtual reality environment.

3. Overall Top Choice

Neither one of the two sensors tested, configured as shown in Chapters II and V, would be ideally suited to personal navigation or immersive virtual reality training environments by themselves. The test data showed inaccuracies and spikes in the outputs that could cause difficulties when integrated into one of those two applications.

Of the two sensors, the 3DM-GX1 was better suited for both personal navigation and immersive virtual reality training because of the improved accuracy in highly dynamic motions, though it would still have significant problems. The yaw orientation accuracy, the tendency to drift, and the discontinuities in the accuracy on impact are all problems that would have to be dealt with as a part of the integration process. A more effective solution would be some combination of the two sensors. The 3DM-GX1 could

provide accurate short-term orientation information, whereas the 3DM-GX3 would be able to provide long-term stationary information and help to remove drift from the 3DM-GX1.

C. RECOMMENDED FUTURE WORK

Although the primary goals that were established were met, there is still much work that can be done on this topic. First and foremost would be to remove any accuracy errors that were due to the data collection equipment and not the sensor's measurement. The largest source of test error was introduced by the timing, especially attempting to resolve the 3DM-GX3 time. For some reason this sensor had much more variability in the timing and required much more interaction to attempt to remove the impact of timing errors in the data analysis. A more accurate method of resolving the sensor time to the encoder time would significantly strengthen the accuracy of the test results and should be one of the primary focuses of any follow-on work.

Another area of work would be to run more tests on the 3DM-GX1 and 3DM-GX3 in order to more fully characterize the accuracies. Multiple "impact and hold" tests in a row, spaced out by a second or two, could be run to more realistically simulate a walking motion. Longer tests could be run to determine how far the 3DM-GX1 drifts after dynamic motions or to characterize how long it takes for the 3DM-GX3 error to return to near zero. Also, other features of the 3DM-GX3 could be enabled to see if any other data provided better dynamic accuracy. Finally, some of the interesting anomalies could be followed up and investigated more closely, such as the 3DM-GX3 semi-static yaw tests or the discontinuities that occur in both sensors' accuracy on impact.

Only two sensors made by the same manufacturer were tested, but there are a number of MARG sensors made by many different manufacturers that could be tested. Future work could focus on using the test apparatus developed in this thesis to test other sensors and compare them to one another in a controlled and systematic manner.

Due to the limitations in generating adequate truth data for angular velocity and acceleration noted in Chapter V, only the sensors' orientation was tested in this thesis. Future work could focus on optimally estimating the encoder data using a Weiner or

Kalman filter and then again attempting to get the angular velocity and acceleration from the filtered encoder data. Alternatively, another source of truth data could be added to the test apparatus. A fiber optic gyro or some other accurate gyroscope could be attached to the end of the shaft that the pendulum rotates about. This instrumented data would immediately provide the angular velocity, and using a combination of these two truth sources makes it more likely that an accurate calculation of the acceleration would be obtained. Thus, it would be feasible to measure the accuracies of the sensors' raw outputs of angular velocity and acceleration as well as the processed orientation data, providing more insight into the capabilities of the MARG sensors under test.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. LABVIEW DIAGRAMS

This appendix contains details on many of the LabVIEW virtual instruments that were presented in the thesis. In Figures 124–129, the subVIs that were used to interface with the MicroStrain sensors via the serial port are shown. This was done in three steps: first the command was sent, then the serial record was read, and finally the serial record was converted for use by Loop 2 of the target VI.

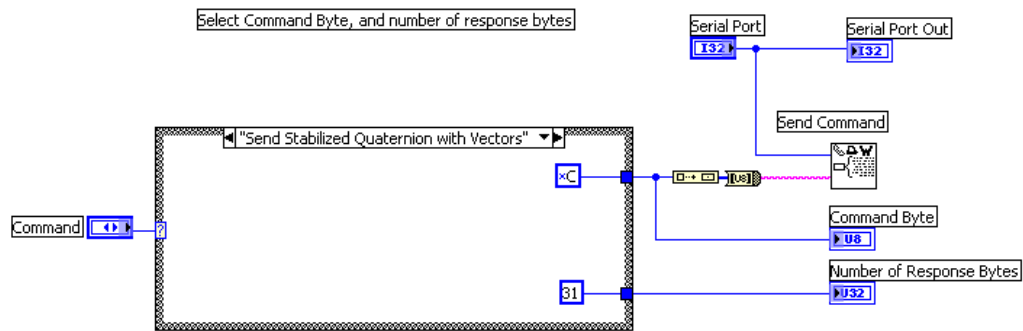


Figure 124. SubVI “Send 3DM-G Cmd.vi” Used in 3DM-GX1 Loop 2.

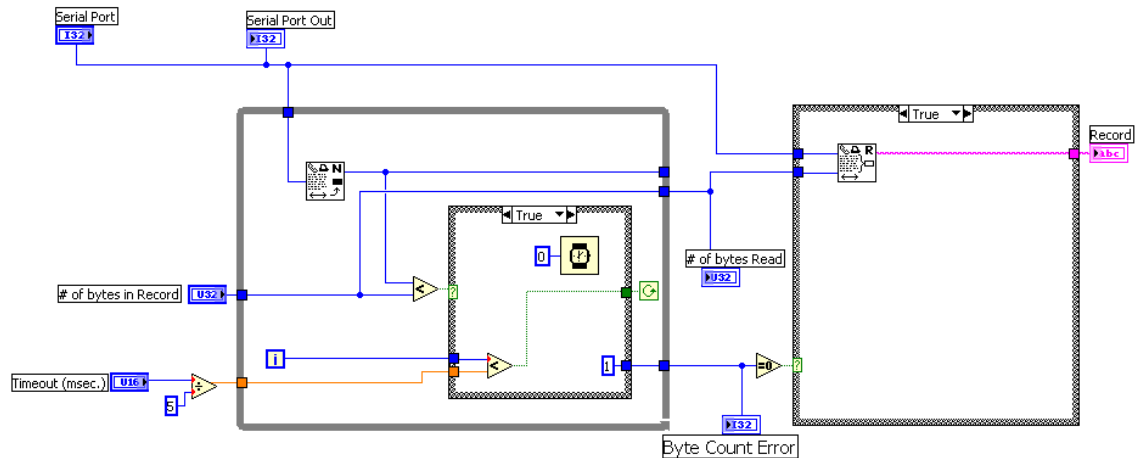


Figure 125. SubVI “Get Serial Record.vi” Used in 3DM-GX1 Loop 2.

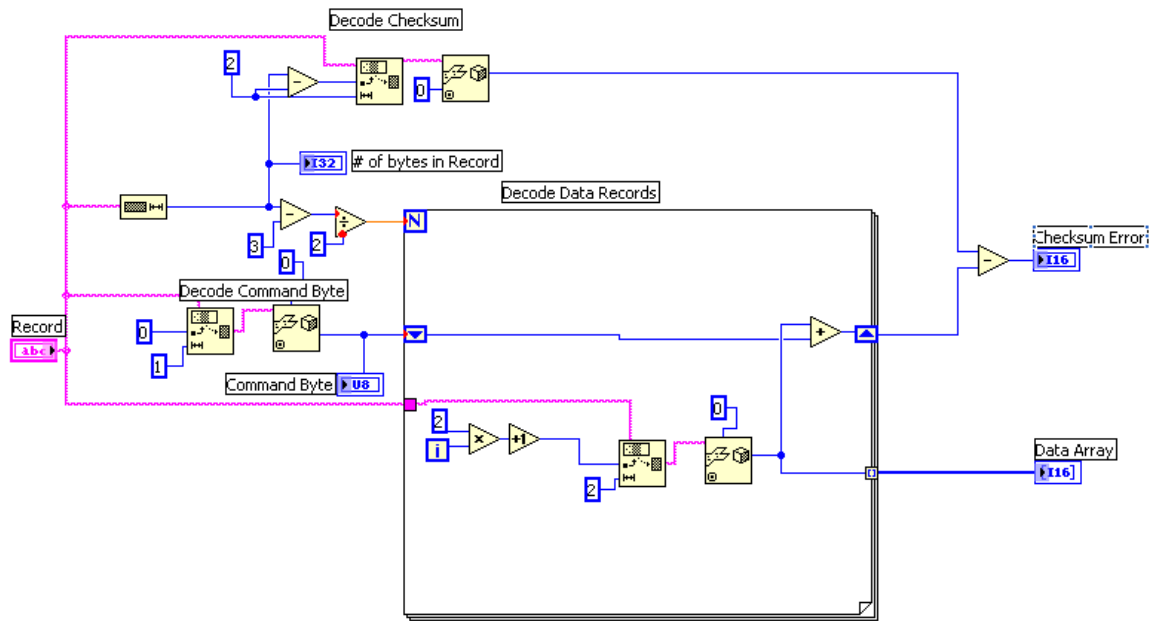


Figure 126. SubVI “Decode 3DM-G Record.vi” Used in 3DM-GX1 Loop 2.

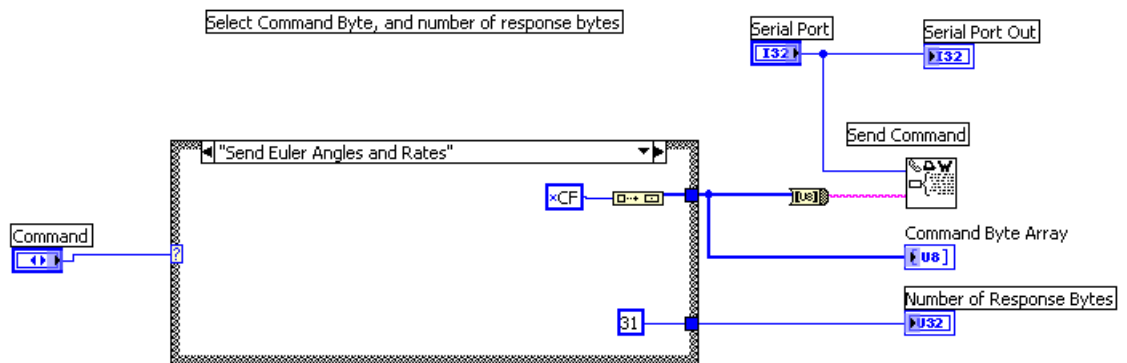


Figure 127. SubVI “Send 3DM-GX3 Cmd.vi” Used in 3DM-GX3 Loop 2.

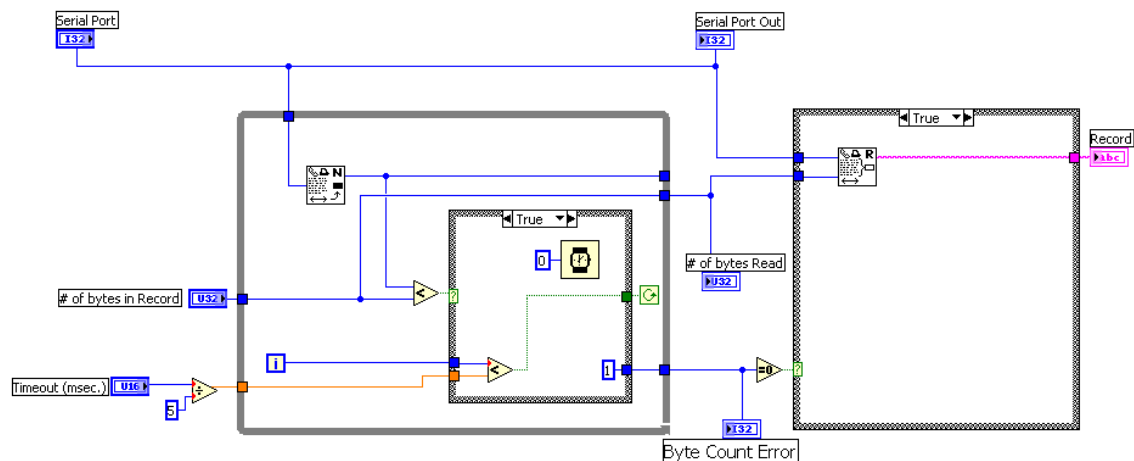


Figure 128. SubVI “Get 3DM-GX3 Serial Record.vi” Used in 3DM-GX3 Loop 2.

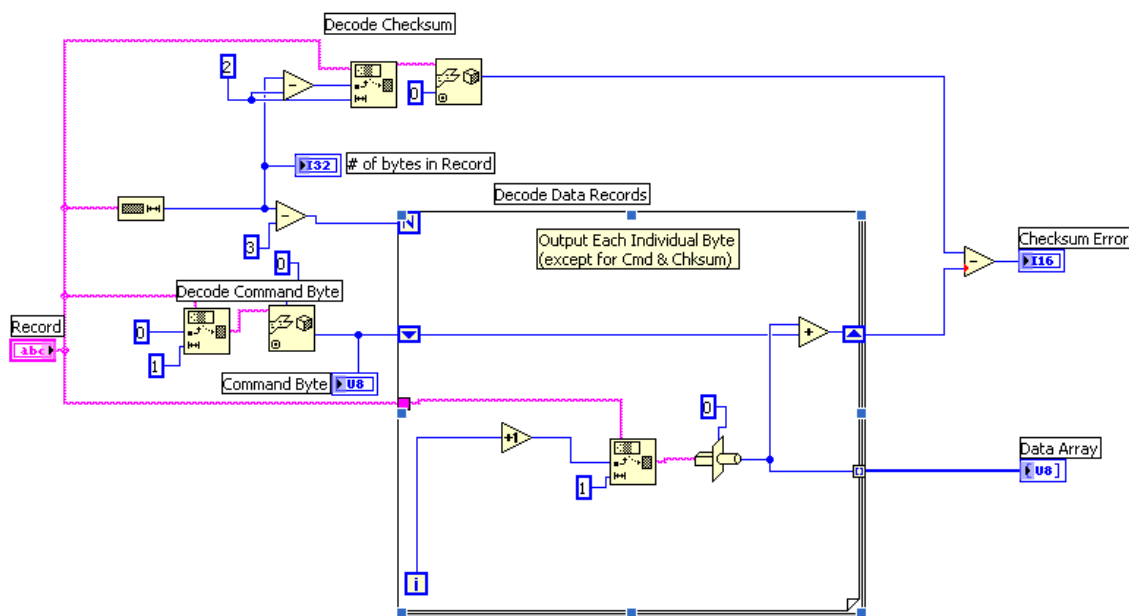
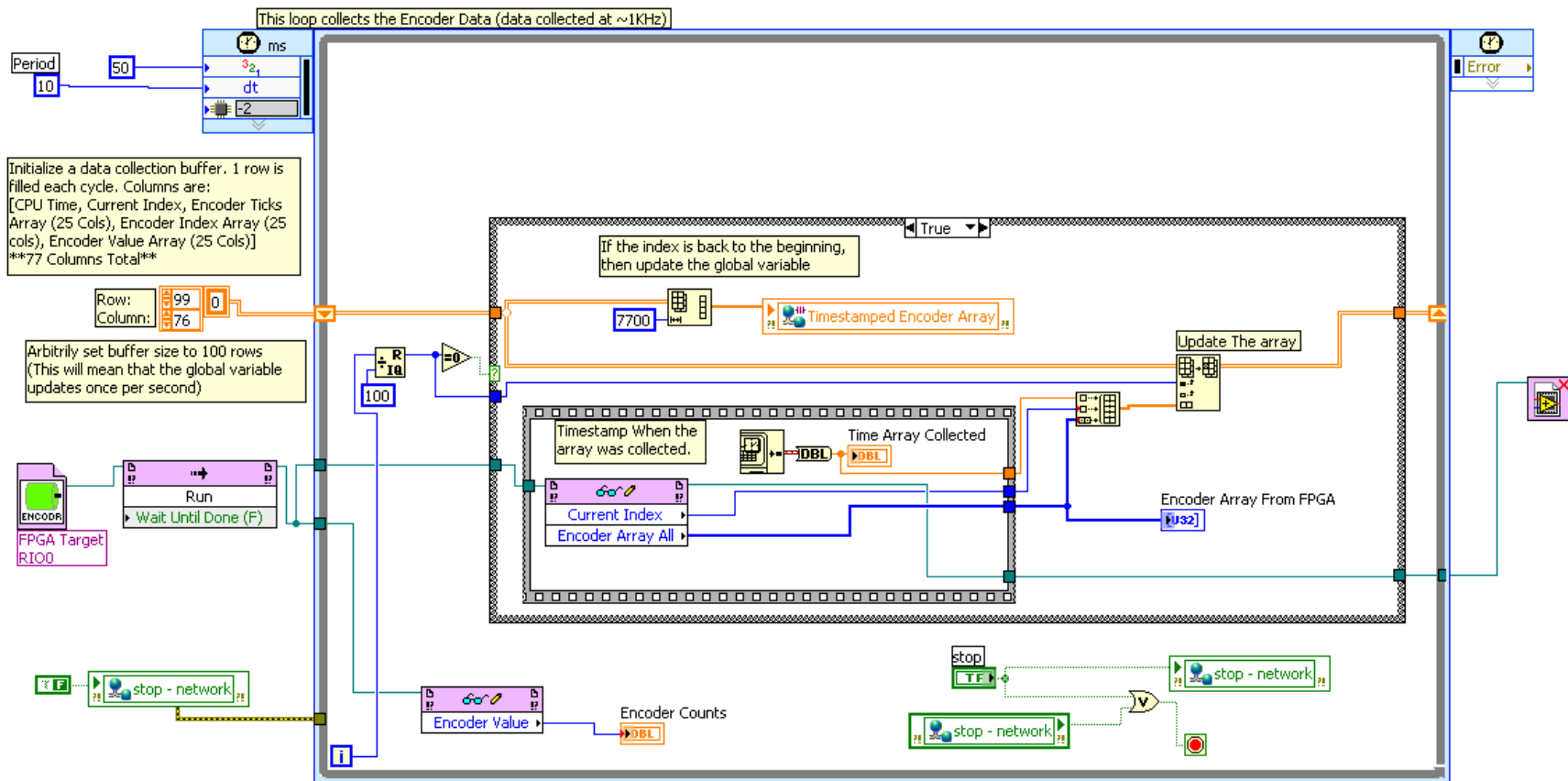


Figure 129. SubVI “Decode 3DM-GX3 Record.vi” Used in 3DM-GX3 Loop 2.

In Figures 130 and 131, the VI used to pull the 3DM-GX1 data is shown in two parts, loop 1 and loop 2. Figure 132 has the modified loop 2 that was used to gather the data from the 3DM-GX1 in continuous mode. Finally, two of the other data options that were available with the 3DM-GX3 are shown in Figures 133 and 134.



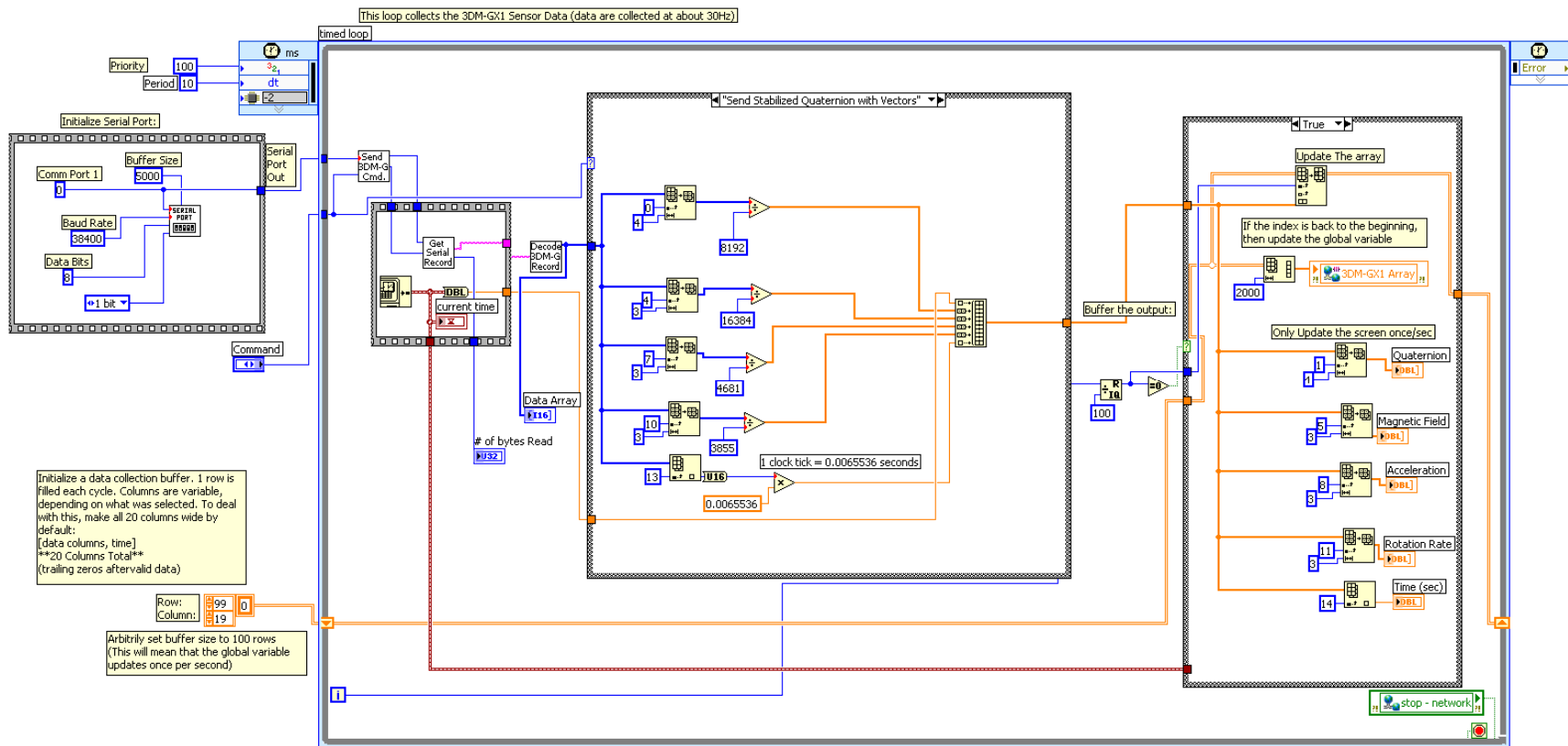


Figure 131. LabVIEW VI for 3DM-GX1, Polled Mode, Loop 2.

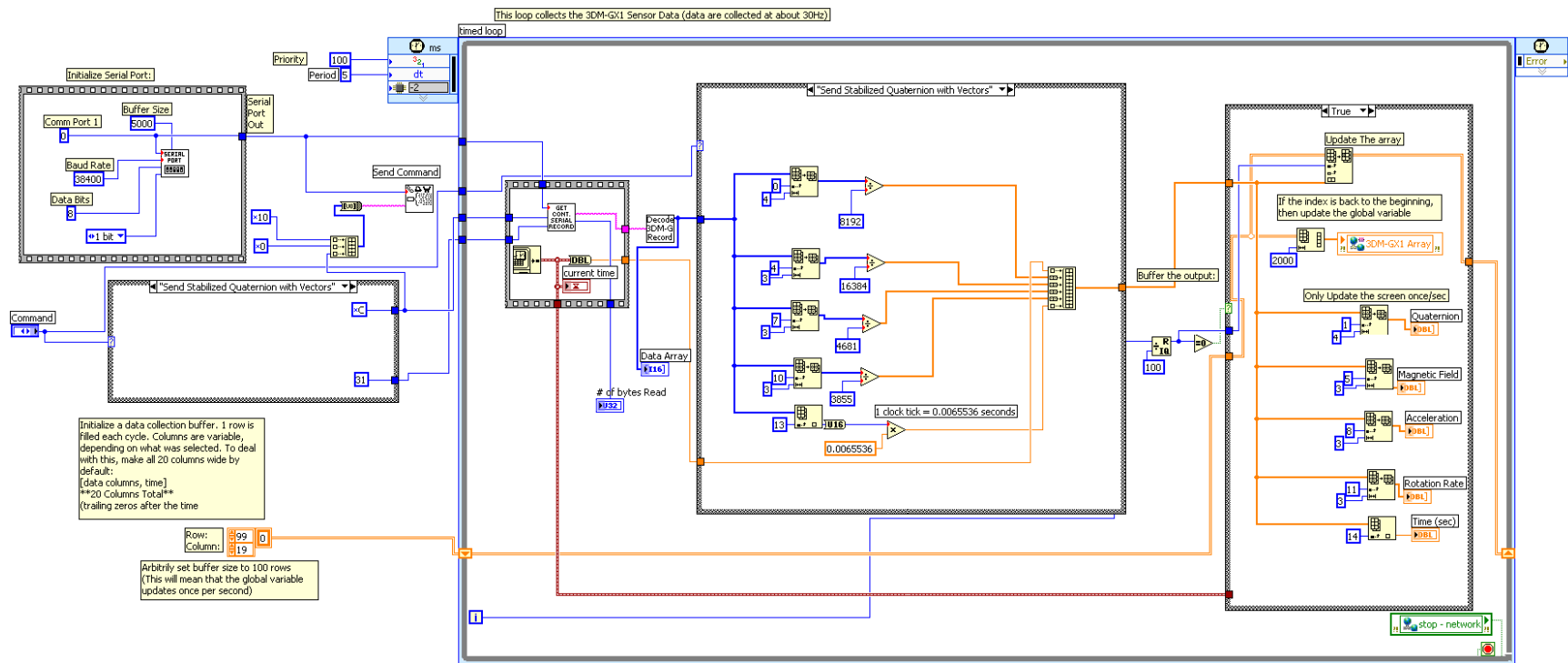


Figure 132. LabVIEW VI for 3DM-GX1, Continuous Mode, Loop 2.

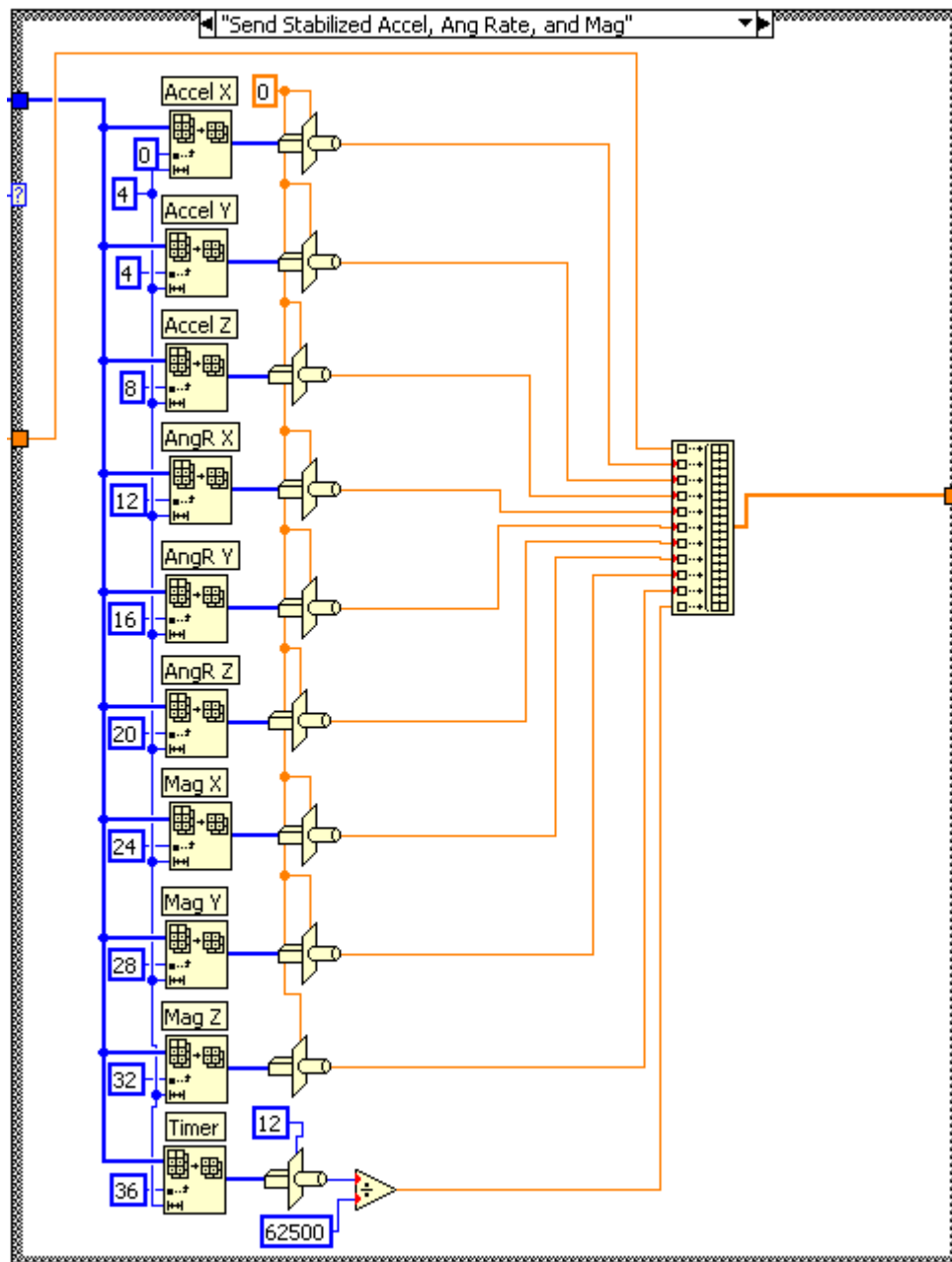


Figure 133. 3DM-GX3, Loop 2, Raw Sensor Measurements Selection.

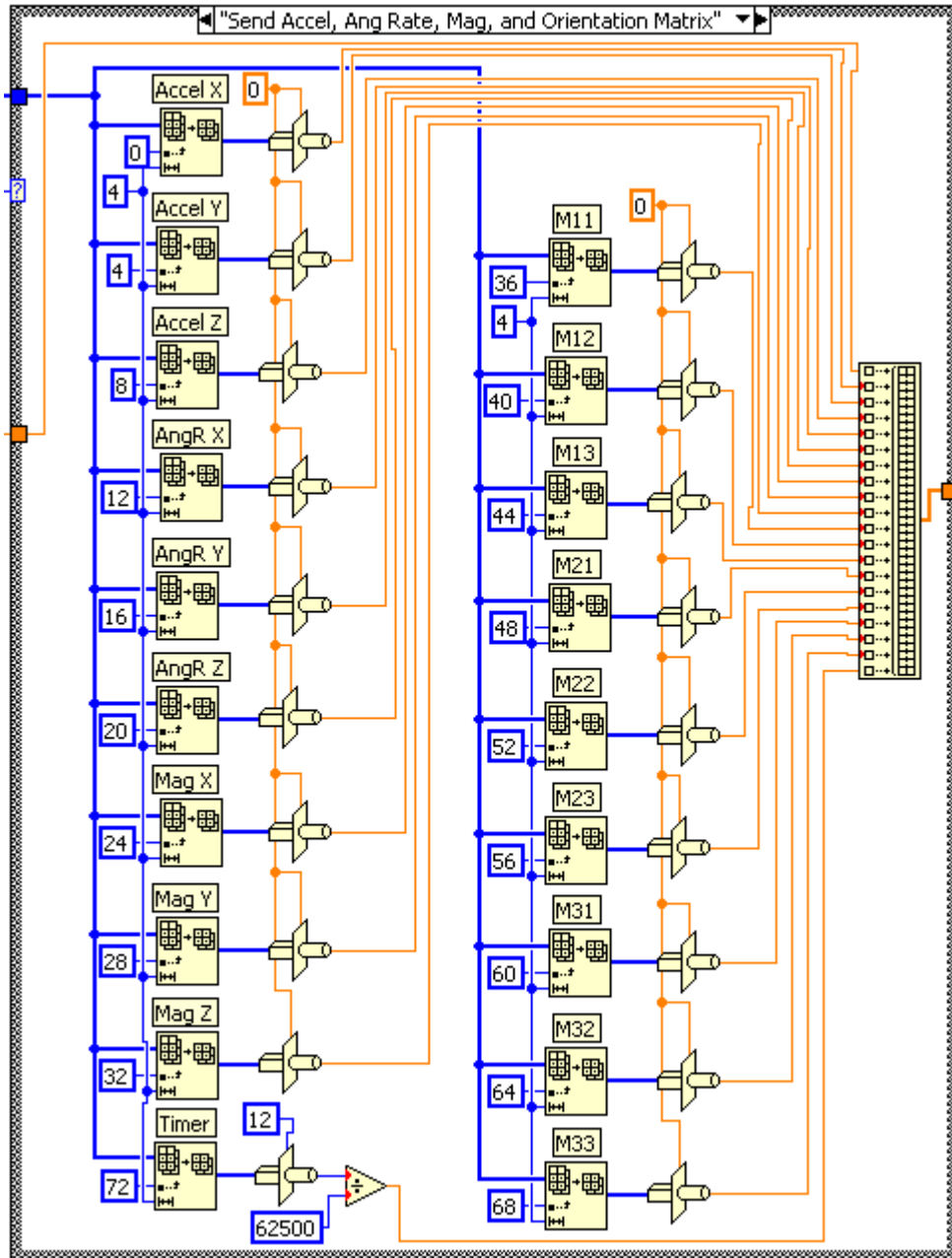


Figure 134. 3DM-GX3, Loop 2, Raw Sensor Measurements and Orientation Matrix.

APPENDIX B. MATLAB CODE

This appendix contains the primary MATLAB code that was used in this thesis. First, selected functions from the code that supported the GUI are presented, since not every function in MEMS_Test.m contained useful code. After this, each subsequent section contains another function that was instrumental in the data analysis portion of the thesis.

A. MEMS_TEST.M, SELECTED FUNCTIONS

```
function varargout = MEMS_Test(varargin)
% MEMS_TEST M-file for MEMS_Test.fig
%     MEMS_Test should be run from the command line. It opens the GUI
%     which enables analysis of the encoder, 3DM-GX1, and 3DM-GX3 data
%     collected by the LabVIEW virtual instrument
%
%     There is no other appropriate way to run or call this GUI
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Last Modified by GUIDE v2.5 27-Apr-2010 10:25:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @MEMS_Test_OpeningFcn, ...
                  'gui_OutputFcn',  @MEMS_Test_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before MEMS_Test is made visible.
function MEMS_Test_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to MEMS_Test (see VARARGIN)

% Choose default command line output for MEMS_Test
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%create a cell variable to contain the name of the variables to save
```

```

setappdata(handles.Main,'Save_Vars',{});

% UIWAIT makes MEMS_Test wait for user response (see UIRESUME)
% uiwait(handles.Main);

% --- Outputs from this function are returned to the command line.
function varargout = MEMS_Test_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit_Encoder_File_Callback(hObject, eventdata, handles)
% hObject handle to edit_Encoder_File (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Encoder_File as text
% str2double(get(hObject,'String')) returns contents of edit_Encoder_File as a
double

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on button press in push_Browse_Encoder.
function push_Browse_Encoder_Callback(hObject, eventdata, handles)
% hObject handle to push_Browse_Encoder (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

%this function prompts the user to enter a file for the encoder

current_file = get(handles.edit_Encoder_File,'String');
if strcmp(current_file, 'Encoder LabView Data File') == 1
    [file path] = uigetfile('*.txt', 'Select Encoder Data'); %first time
else %use the current path at the default location:
    [file path] = uigetfile('*.txt', 'Select Encoder Data',current_file);
end

if file == 0
    %action was canx'd, don't change anything
    return
else
    set(handles.edit_Encoder_File,'String',[path file]);
end

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

function edit_Sensor_File_Callback(hObject, eventdata, handles)
% hObject handle to edit_Sensor_File (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB

```

```

% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Sensor_File as text
%        str2double(get(hObject,'String')) returns contents of edit_Sensor_File as a
double

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{' ':'Data Not Updated';''});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on button press in push_Browse_Sensor.
function push_Browse_Sensor_Callback(hObject, eventdata, handles)
% hObject      handle to push_Browse_Sensor (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

%this function prompts the user to enter a file for the sensor

current_file = get(handles.edit_Sensor_File,'String');
if strcmp(current_file, 'Sensor LabView Data File') == 1
    [file path] = uigetfile('*.txt', 'Select Sensor Data'); %first time
else %use the current path at the default location:
    [file path] = uigetfile('*.txt', 'Select Sensor Data',current_file);
end

if file == 0
    %action was canx'd, don't change anything
    return
else
    set(handles.edit_Sensor_File,'String',[path file]);
end

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{' ':'Data Not Updated';''});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on selection change in pop_Selected_Sensor.
function pop_Selected_Sensor_Callback(hObject, eventdata, handles)
% hObject      handle to pop_Selected_Sensor (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns pop_Selected_Sensor contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from pop_Selected_Sensor

contents = get(handles.pop_Selected_Sensor,'String');
selected_sensor = contents{get(hObject,'Value')};

%When a sensor is selected, populate the "Sensor_Data_Collected" popup menu
%with the appropriate values/options (also, clean up the 'calling' popmenu)
switch selected_sensor
case '3DM-GX1'
    set(handles.pop_Sensor_Data_Collected,'Enable','on');
    set(handles.pop_Sensor_Data_Collected,'Value',2);
    set(handles.pop_Sensor_Data_Collected,'String',...
        {'Select Sensor Data Pulled';'Stabilized Quaternions & Vectors';'Serial
Number'});
    set(handles.pop_Selected_Sensor,'String',{'3DM-GX1';'3DM-GX3'});
    set(handles.pop_Selected_Sensor,'Value',1);

    set(handles.edit_Title_Line,'String','3DM-GX1');

```

```

    case '3DM-GX3'
        set(handles.pop_Sensor_Data_Collected,'Enable','on');
        set(handles.pop_Sensor_Data_Collected,'Value',2);
        set(handles.pop_Sensor_Data_Collected,'String',...
            {'Select Sensor Data Pulled';'Euler Angles and Rates';'Euler Angles
Only';'Serial Number'});
        set(handles.pop_Selected_Sensor,'String',{'3DM-GX1';'3DM-GX3'});
        set(handles.pop_Selected_Sensor,'Value',2);

        set(handles.edit_Title_Line,'String','3DM-GX3');
    end

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on selection change in pop_Sensor_Data_Collected.
function pop_Sensor_Data_Collected_Callback(hObject, eventdata, handles)
% hObject    handle to pop_Sensor_Data_Collected (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns pop_Sensor_Data_Collected contents as
cell array
%          contents{get(hObject,'Value')} returns selected item from
pop_Sensor_Data_Collected

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% -----
function menu_Quit_Callback(hObject, eventdata, handles)
% hObject    handle to menu_Quit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%close the gui
close

function edit_Rotation_Threshold_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Rotation_Threshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Rotation_Threshold as text
%          str2double(get(hObject,'String')) returns contents of edit_Rotation_Threshold as
a double

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

```

```

% --- Executes during object creation, after setting all properties.
function edit_Rotation_Threshold_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Rotation_Threshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in check_Avg_Encoder_Times.
function check_Avg_Encoder_Times_Callback(hObject, eventdata, handles)
% hObject    handle to check_Avg_Encoder_Times (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of check_Avg_Encoder_Times

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'':'Data Not Updated':''});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on button press in check_Adjust_Sensor_Times.
function check_Adjust_Sensor_Times_Callback(hObject, eventdata, handles)
% hObject    handle to check_Adjust_Sensor_Times (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of check_Adjust_Sensor_Times

%only do one type of time fix at a time
if get(handles.check_Manual_Time_Adjust,'Value') == 1
    set(handles.check_Manual_Time_Adjust,'Value',0);%uncheck the "Peak Align" time
adjust"
else
    set(handles.check_Manual_Time_Adjust,'Value',1);%check the "Peak Align" time adjust"
end

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'':'Data Not Updated':''});
%disable saving changed data
set(handles.push_Save,'Enable','off');

function edit_Zero_Motion_Time_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Zero_Motion_Time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Zero_Motion_Time as text
%         str2double(get(hObject,'String')) returns contents of edit_Zero_Motion_Time as a
double

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'':'Data Not Updated':''});
%disable saving changed data

```

```

set(handles.push_Save, 'Enable', 'off');

% --- Executes during object creation, after setting all properties.
function edit_Zero_Motion_Time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Zero_Motion_Time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

% --- Executes on button press in radio_Pitch.
function radio_Pitch_Callback(hObject, eventdata, handles)
% hObject    handle to radio_Pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject, 'Value') returns toggle state of radio_Pitch

%only allow 1 selection
set(handles.radio_Roll, 'Value', 0);
set(handles.radio_Pitch, 'Value', 1);
set(handles.radio_Yaw, 'Value', 0);

%preset the plotting selections based on "Pitch" selected
%1
set(handles.radio_1_Roll, 'Value', 0);
set(handles.radio_1_Pitch, 'Value', 1);
set(handles.radio_1_Yaw, 'Value', 0);
%2
set(handles.radio_2_Roll, 'Value', 0);
set(handles.radio_2_Pitch, 'Value', 1);
set(handles.radio_2_Yaw, 'Value', 0);
%error plot
set(handles.check_Error_Plot_Roll, 'Value', 0);
set(handles.check_Error_Plot_Pitch, 'Value', 1);
set(handles.check_Error_Plot_Yaw, 'Value', 0);

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated, 'BackgroundColor', [.961, .922, .922]);
set(handles.text_Data_Updated, 'ForegroundColor', [1, 0, 0]);
set(handles.text_Data_Updated, 'String', {''; 'Data Not Updated'; ''});
%disable saving changed data
set(handles.push_Save, 'Enable', 'off');

% --- Executes on button press in radio_Yaw.
function radio_Yaw_Callback(hObject, eventdata, handles)
% hObject    handle to radio_Yaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject, 'Value') returns toggle state of radio_Yaw

%only allow 1 selection
set(handles.radio_Roll, 'Value', 0);
set(handles.radio_Pitch, 'Value', 0);
set(handles.radio_Yaw, 'Value', 1);

%preset the plotting selections based on "Roll" selected
%1
set(handles.radio_1_Roll, 'Value', 0);
set(handles.radio_1_Pitch, 'Value', 0);

```

```

set(handles.radio_1_Yaw,'Value',1);
%2
set(handles.radio_2_Roll,'Value',0);
set(handles.radio_2_Pitch,'Value',0);
set(handles.radio_2_Yaw,'Value',1);
%error plot
set(handles.check_Error_Plot_Roll,'Value',0);
set(handles.check_Error_Plot_Pitch,'Value',0);
set(handles.check_Error_Plot_Yaw,'Value',1);

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on button press in radio_Roll.
function radio_Roll_Callback(hObject, eventdata, handles)
% hObject    handle to radio_Roll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_Roll

%only allow 1 selection
set(handles.radio_Roll,'Value',1);
set(handles.radio_Pitch,'Value',0);
set(handles.radio_Yaw,'Value',0);

%preset the plotting selections based on "Roll" selected
%1
set(handles.radio_1_Roll,'Value',1);
set(handles.radio_1_Pitch,'Value',0);
set(handles.radio_1_Yaw,'Value',0);
%2
set(handles.radio_2_Roll,'Value',1);
set(handles.radio_2_Pitch,'Value',0);
set(handles.radio_2_Yaw,'Value',0);
%error plot
set(handles.check_Error_Plot_Roll,'Value',1);
set(handles.check_Error_Plot_Pitch,'Value',0);
set(handles.check_Error_Plot_Yaw,'Value',0);

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'Data Not Updated'});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes on button press in push_Run.
function push_Run_Callback(hObject, eventdata, handles)
% hObject    handle to push_Run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% This function pulls the files from the GUI, then opens them in accordance
% with the GUI's settings. The final data are stored to the appdata of the
% background, MEMS_Test

% The data stored to the appdata are:
% Time_Encoder = [CPU Time, common elapsed time, local delta time]
% Time_Sensor = [CPU Time, common elapsed time, local delta time]
% Encoder_Angle = [elapsed time, deg, rad]

```

```

% Encoder_TSPI_w = [elapsed time, deg/s, rad/s]
% Encoder_TSPI_alpha = [elapsed time, deg/s^2, rad/s^2]
% Sensor_RPY_rad = [elapsed time, roll, pitch, yaw]; in radians
% Sensor_RPY_deg = [elapsed time, roll, pitch, yaw]; in degrees

% FIRST GET THE FILES AND ENSURE THEY EXIST:
en_filename = get(handles.edit_Encoder_File,'String');
sen_filename = get(handles.edit_Sensor_File,'String');

if exist(en_filename)==0 || exist(sen_filename)==0
    beep
    errordlg({'Invalid File Name';'One or both of the selected files does not exist';'Fix it!'},'File Error!')
    return
end

%make sure the file names are not the exact same
if strcmp(en_filename, sen_filename) == 1
    beep
    errordlg({'Invalid File Name';'You can't have the same file for the encoder and sensor';'Fix it!'},'File Error!')
    return
end

% NEXT MAKE SURE A SENSOR AND DATA TYPE HAVE BEEN SELECTED
sensor_string = get(handles.pop_Selected_Sensor,'String');
sensor = sensor_string{get(handles.pop_Selected_Sensor,'Value')};
sensor_string = get(handles.pop_Sensor_Data_Collected,'String');
data_selection = sensor_string{get(handles.pop_Sensor_Data_Collected,'Value')};

if strcmp(sensor,'Select MEMS Sensor')==1 || ...
    strcmp(data_selection,'Select Sensor Data Pulled')==1
    beep
    errordlg({'Invalid Sensor Selection';'Make sure you have selected a sensor';...
        'and the type of data you want from it!'},'Sensor Selection Error')
    return
end

%import the variables to be saved (& added to)
Save_Vars = getappdata(handles.Main,'Save_Vars');

%% With all of the settings good, begin the real work:
c = clock;
m = msgbox({'Began Execution
at:',[sprintf('%2.0f',c(4)),':',sprintf('%2.0f',c(5)),':',sprintf('%2.1f',c(6))]},'Executing');

% Set some output params to 'disabled' until they are updated:
set(handles.text_Avg_Error,'Enable','off');
set(handles.text_Time_Align_Error,'Enable','off');

% NEXT GET ALL OF THE SETUP PARAMETERS FROM THE GUI
% Encoder Parameters
block_size = str2num(get(handles.edit_Encoder_Block_Size,'String'));
Fs = str2num(get(handles.edit_Encoder_Sampling_Freq,'String'));
encoder_frame_period = 1/Fs;
threshold = str2num(get(handles.edit_Rotation_Threshold,'String'));
r_pend = str2num(get(handles.edit_Rotation_Radius,'String'));

% Sensor Parameters
%(these indicate what the encoder angle represents on the sensor)
roll_checked = get(handles.radio_Roll,'Value');
pitch_checked = get(handles.radio_Pitch,'Value');
yaw_checked = get(handles.radio_Yaw,'Value');
if roll_checked == 1
    sensor_orientation = 'Roll';
elseif pitch_checked==1
    sensor_orientation = 'Pitch';
else

```

```

        sensor_orientation = 'Yaw';
    end

    sensor_peak_time_align_checked = get(handles.check_Adjust_Sensor_Times, 'Value');
    if sensor_peak_time_align_checked == 1
        sensor_peak_time_align_checked = true;
    else
        sensor_peak_time_align_checked = false;
    end

    %Analysis Parameters
    steady_state = str2num(get(handles.edit_Zero_Motion_Time, 'String')); %assumes first X
    seconds of file are starting point of sensor

    %% READ IN THE ENCODER FILE:
    [time_en encoder_angle] =
    Read_Encoder_Data_File(en_filename, block_size, encoder_frame_period);

    %% READ IN THE SENSOR DATA
    raw_data = textread(sen_filename, '%f'); %read as double array

    %Note: all sensor data files are arranged into 20 columns of data.
    raw_resaped = reshape(raw_data, 20, length(raw_data)/20)';

    %find any duplicate lines and remove them:
    t = raw_resaped(:, 1); %CPU Time is the first column
    t(:, 2) = [0; t(2:end, 1) - t(1:end-1, 1)];
    non_zero_i = t(:, 2) ~ zeros(length(t(:, 2)), 1);
    time_3DM = t(non_zero_i, :);

    %build a new data array with no duplicate lines:
    data = raw_resaped(non_zero_i, :);

    %set up an elapsed time & move the time delta column:
    time_3DM(:, 2) = time_3DM(:, 1) - time_3DM(1, 1) * ones(length(time_3DM), 1); %elapsed t
    time_3DM(:, 3) = [0; time_3DM(2:end, 1) - time_3DM(1:end-1, 1)]; %delta t

    %Note: the sensor data read in will vary depending on the selected data. It
    %will also vary for sensor types
    switch sensor
        case '3DM-GX1'
            %For the -GX1, pull the appropriate data:
            switch data_selection
                case 'Stabilized Quaternions & Vectors'
                    %Break out the data by type:
                    quat = data(:, 2:5); %quaternions
                    mag = data(:, 6:8); %magnetic field (gauss)
                    accel = data(:, 9:11); %acceleration (g)
                    angr = data(:, 12:14); %angular rate (rad/s)
                    tick_time = data(:, 15);

                    %Convert Quaternions to Roll/Pitch/Yaw
                    [rpy_rad rpy_deg] = quat2rpy(quat);

                    %scale the roll term by -1, since the pendulum data and +/-
                    %roll convention are opposite:
                    rpy_rad(:, 1) = -1.*rpy_rad(:, 1);
                    rpy_deg(:, 1) = -1.*rpy_deg(:, 1);

                case 'Serial Number'
                    errordlg('Not Yet Configured')
                    return
                    %pull these
            otherwise
                errordlg('Internal Code Error -- Invalid Data Selection Allowed')
                return
            end %switch data_selection
    end

```

```

case '3DM-GX3'
    %For the -GX3, pull the appropriate data:
    switch data_selection
        case 'Euler Angles and Rates'
            rpy_rad(:,1:3) = data(:,2:4); %roll,pitch,yaw
            rpy_deg = rpy_rad.*180/pi;
            angr = data(:,5:7); %angular rates (rad/s)
            tick_time = data(:,8);

            %scale the roll term by -1, since the pendulum data and +/-
            %roll convention are opposite:
            rpy_rad(:,1) = -1.*rpy_rad(:,1);
            rpy_deg(:,1) = -1.*rpy_deg(:,1);

        case 'Euler Angles Only'
            rpy_rad(:,1:3) = data(:,2:4); %roll,pitch,yaw
            rpy_deg = rpy_rad.*180/pi;
            tick_time = data(:,5);

            %scale the roll term by -1, since the pendulum data and +/-
            %roll convention are opposite:
            rpy_rad(:,1) = -1.*rpy_rad(:,1);
            rpy_deg(:,1) = -1.*rpy_deg(:,1);

        case 'Serial Number'
            errorordlg('Not Yet Configured')
            return
            %pull these
        otherwise
            errorordlg('Internal Code Error -- Invalid Data Selection Allowed')
            return
    end %switch data_selection
end %switch sensor

%% SYNC THE TIMES TO A COMMON ELAPSED TIME:
if time_en(1,1)<= time_3DM(1,1)
    sync_index = find(time_en(:,1)> time_3DM(1,1),1,'first');
    sync_time = time_en(sync_index,1);
else
    sync_index = find(time_3DM(:,1)> time_en(1,1),1,'first');
    sync_time = time_3DM(sync_index,1);
end

time_en(:,2) = time_en(:,1)-sync_time*ones(length(time_en),1); %common elapsed time
time_3DM(:,2) = time_3DM(:,1)-sync_time*ones(length(time_3DM),1); %common elapsed time

%% IF SELECTED, THRESHOLD THE ENCODER DATA TO THE SELECTED VALUE
if get(handles.check_Use_Max_Rotation_Thresh,'Value')==1
    [new_time_en new_encoder_angle] =
Encoder_Threshold(time_en,encoder_angle,threshold,Fs); %sets max rotation to 'threshold'
deg/sec
    clear encoder_angle time_en

    time_en = new_time_en;
    encoder_angle = new_encoder_angle;

    clear new_time new_encoder_angle
end

%% CONVERT ENCODER DATA TO TSPI
%NOTE: This also adjusts the encoder start position to Zero Degrees.
[encoder_angle,w,ang_accel,time_en,accel_xyz,g] = ...
encoder_TSPI(encoder_angle, time_en, r_pend, steady_state,sensor_orientation);
%accel_xyz returns the TSPI for x, y, & z body accelerations in DEGREES ONLY

%% CALCULATE THE FFT OF THE ENCODER ANGLE, ANGULAR VELOCITY, AND ANG. ACCEL
fft_en = fft(encoder_angle(:,2));
fft_w = fft(w(:,2));

```

```

fft_alpha = fft(ang_accel(:,2));

%% RUN THE ACCURACY ANALYSIS & SAVE RESULTS TO THE GUI'S APPDATA
%NOTE: Different Accuracy Analysis will be done for the different sensors
%therefore, the APPDATA may not contain all of the variables each time

%Identify the steady state time index for the sensor data
tare_index = find(time_3DM(:,2)>steady_state, 1, 'first');

switch sensor
    case '3DM-GX1'
        switch data_selection
            case 'Stabilized Quaternions & Vectors'
                %Identify which axis will be tested (this will result in a
                %lot of extra/duplicate code, but it will work)
                switch sensor_orientation
                    case 'Roll'
                        %Identify the initial orientation error
                        setup_roll_err = median([rpy_deg(1:tare_index,1),
rpy_rad(1:tare_index,1)]); %[deg rad]

                        if sensor_peak_time_align_checked
                            % Adjust Microstrain Sensor Time To Account for Misalignment
w/ cRIO

                            %'Fix' the sensor time by lining it up with the encoder peaks
                            [time_3DM time_error] = sensor_time_align(time_en, time_3DM,
encoder_angle, rpy_deg, sensor_orientation);
                            set(handles.text_Time_Align_Error, 'Enable', 'on');
                        else
                            %manually adjust by the time input on the GUI
                            time_error =
str2num(get(handles.edit_Manual_Time_Adjust, 'String')).*.001; %convert to milliseconds
                            time_3DM(:,1:2) = time_3DM(:,1:2) - time_error;
                            set(handles.text_Time_Align_Error, 'Enable', 'on');
                        end

                        %interpolate the encoder data to the sensor times
                        encoder_interp(:,2) =
interp1(time_en(:,2), encoder_angle(:,2), time_3DM(:,2)); %degree, linear interp
                        encoder_interp(:,3) =
interp1(time_en(:,2), encoder_angle(:,3), time_3DM(:,2)); %rad, linear interp

                        %replace any NaN with zeros
                        encoder_interp(isnan(encoder_interp)) = 0;

                        %adjust the appropriate rpy column to account for initial
alignment errors
                        adjusted_rpy_deg = rpy_deg;
                        adjusted_rpy_rad = rpy_rad;
                        adjusted_rpy_deg(:,1) = rpy_deg(:,1) - setup_roll_err(1,1);
                        adjusted_rpy_rad(:,1) = rpy_rad(:,1) - setup_roll_err(1,2);

                        %crop the data to start at common elapsed time = 0,
                        index_pos_time = time_3DM(:,2)>=0;
                        %then take the difference/calculate error
                        roll_error = [time_3DM(index_pos_time,2), ...
adjusted_rpy_deg(index_pos_time,1) -
encoder_interp(index_pos_time,2), ...
adjusted_rpy_rad(index_pos_time,1) -
encoder_interp(index_pos_time,3)];

                        %update the screen data: (only count the error from after the
'still' time)
                        set(handles.text_Avg_Error, 'Enable', 'on');

set(handles.text_Avg_Error, 'String', num2str(sqrt(mean(roll_error(tare_index:end,2).^2)))
;
                        set(handles.text_Time_Align_Error, 'String', num2str(time_error));

```

```

        %update the appdata
        setappdata(handles.Main,'Error_Roll',roll_error);
        setappdata(handles.Main,'Adjusted_rpy_deg',adjusted_rpy_deg);
        setappdata(handles.Main,'Adjusted_rpy_rad',adjusted_rpy_rad);
        setappdata(handles.Main,'Interpolated_Encoder',encoder_interp);

        %update the save variable
        Save_Vars{end+1} = 'Error_Roll';
        Save_Vars{end+1} = 'Adjusted_rpy_deg';
        Save_Vars{end+1} = 'Adjusted_rpy_rad';
        Save_Vars{end+1} = 'Interpolated_Encoder';
    case 'Pitch'
        %Identify the initial orientation error
        setup_pitch_err = median([rpy_deg(1:tare_index,2),
rpy_rad(1:tare_index,2)]); %[deg rad]

        if sensor_peak_time_align_checked
            % Adjust Microstrain Sensor Time To Account for Misalignment
w/ cRIO

            %'Fix' the sensor time by lining it up with the encoder peaks
            [time_3DM time_error] = sensor_time_align(time_en, time_3DM,
encoder_angle, rpy_deg,sensor_orientation);
            set(handles.text_Time_Align_Error,'Enable','on');
        else
            %manually adjust by the time input on the GUI
            time_error =
str2num(get(handles.edit_Manual_Time_Adjust,'String')).*.001; %convert to milliseconds
            time_3DM(:,1:2) = time_3DM(:,1:2) - time_error;
            set(handles.text_Time_Align_Error,'Enable','on');
        end

        %interpolate the encoder data to the sensor times
        encoder_interp(:,2) =
interp1(time_en(:,2),encoder_angle(:,2),time_3DM(:,2)); %degree, linear interp
        encoder_interp(:,3) =
interp1(time_en(:,2),encoder_angle(:,3),time_3DM(:,2)); %rad, linear interp

        %replace any NaN with zeros
        encoder_interp(isnan(encoder_interp)) = 0;

        %adjust the appropriate rpy column to account for initial
alignment errors
        adjusted_rpy_deg = rpy_deg;
        adjusted_rpy_rad = rpy_rad;
        adjusted_rpy_deg(:,2) = rpy_deg(:,2) - setup_pitch_err(1,1);
        adjusted_rpy_rad(:,2) = rpy_rad(:,2) - setup_pitch_err(1,2);

        %crop the data to start at common elapsed time = 0,
        index_pos_time = time_3DM(:,2)>=0;
        %then take the difference/calculate error
        pitch_error = [time_3DM(index_pos_time,2), ...
            adjusted_rpy_deg(index_pos_time,2) -
encoder_interp(index_pos_time,2),...
            adjusted_rpy_rad(index_pos_time,2) -
encoder_interp(index_pos_time,3)];

        %update the screen data: (only count the error from after the
'still' time)
        set(handles.text_Avg_Error,'Enable','on');

        set(handles.text_Avg_Error,'String',num2str(sqrt(mean(pitch_error(tare_index:end,2).^2)))
);

        set(handles.text_Time_Align_Error,'String',num2str(time_error));

        %update the appdata
        setappdata(handles.Main,'Error_Pitch',pitch_error);
        setappdata(handles.Main,'Adjusted_rpy_deg',adjusted_rpy_deg);
        setappdata(handles.Main,'Adjusted_rpy_rad',adjusted_rpy_rad);
        setappdata(handles.Main,'Interpolated_Encoder',encoder_interp);

```

```

        %update the save variable
        Save_Vars{end+1} = 'Error_Pitch';
        Save_Vars{end+1} = 'Adjusted_rpy_deg';
        Save_Vars{end+1} = 'Adjusted_rpy_rad';
        Save_Vars{end+1} = 'Interpolated_Encoder';

        case 'Yaw'
            %Identify the initial orientation error
            setup_yaw_err = median([rpy_deg(1:tare_index,3),
rpy_rad(1:tare_index,3)]); %[deg rad]

            if sensor_peak_time_align_checked
                % Adjust Microstrain Sensor Time To Account for Misalignment
w/ cRIO

                %'Fix' the sensor time by lining it up with the encoder peaks
                [time_3DM time_error] = sensor_time_align(time_en, time_3DM,
encoder_angle, rpy_deg,sensor_orientation);
                set(handles.text_Time_Align_Error,'Enable','on');
            else
                %manually adjust by the time input on the GUI
                time_error =
str2num(get(handles.edit_Manual_Time_Adjust,'String')).*.001; %convert to milliseconds
                time_3DM(:,1:2) = time_3DM(:,1:2) - time_error;
                set(handles.text_Time_Align_Error,'Enable','on');
            end

            %interpolate the encoder data to the sensor times
            encoder_interp(:,2) =
interp1(time_en(:,2),encoder_angle(:,2),time_3DM(:,2)); %degree, linear interp
            encoder_interp(:,3) =
interp1(time_en(:,2),encoder_angle(:,3),time_3DM(:,2)); %rad, linear interp

            %replace any NaN with zeros
            encoder_interp(isnan(encoder_interp)) = 0;

            %adjust the appropriate rpy column to account for initial
alignment errors
            adjusted_rpy_deg = rpy_deg;
            adjusted_rpy_rad = rpy_rad;
            adjusted_rpy_deg(:,3) = rpy_deg(:,3) - setup_yaw_err(1,1);
            adjusted_rpy_rad(:,3) = rpy_rad(:,3) - setup_yaw_err(1,2);

            %crop the data to start at common elapsed time = 0,
            index_pos_time = time_3DM(:,2)>=0;
            %then take the difference/calculate error
            yaw_error = [time_3DM(index_pos_time,2), ...
                adjusted_rpy_deg(index_pos_time,3) -
encoder_interp(index_pos_time,2),...
                adjusted_rpy_rad(index_pos_time,3) -
encoder_interp(index_pos_time,3)];

            %update the screen data: (only count the error from after the
'still' time)
            set(handles.text_Avg_Error,'Enable','on');

            set(handles.text_Avg_Error,'String',num2str(sqrt(mean(yaw_error(tare_index:end,2).^2))));
            set(handles.text_Time_Align_Error,'String',num2str(time_error));

            %update the appdata
            setappdata(handles.Main,'Error_Yaw',yaw_error);
            setappdata(handles.Main,'Adjusted_rpy_deg',adjusted_rpy_deg);
            setappdata(handles.Main,'Adjusted_rpy_rad',adjusted_rpy_rad);
            setappdata(handles.Main,'Interpolated_Encoder',encoder_interp);

            %update the save variable
            Save_Vars{end+1} = 'Error_Yaw';
            Save_Vars{end+1} = 'Adjusted_rpy_deg';
            Save_Vars{end+1} = 'Adjusted_rpy_rad';

```

```

        Save_Vars{end+1} = 'Interpolated_Encoder';
    end %switch sensor_orientation

    % Update common appdata for this set of data:
    setappdata(handles.Main,'Sensor_rpy_deg',rpy_deg);
    setappdata(handles.Main,'Sensor_rpy_rad',rpy_rad);
    setappdata(handles.Main,'Sensor_angr',angr);
    setappdata(handles.Main,'Tare_Index',tare_index);

    %update the save variable
    Save_Vars{end+1} = 'Sensor_rpy_deg';
    Save_Vars{end+1} = 'Sensor_rpy_rad';
    Save_Vars{end+1} = 'Sensor_angr';
    Save_Vars{end+1} = 'Tare_Index';

    case 'Serial Number'
        beep
        errordlg('Not Yet Configured')
        return
        %pull these
    otherwise
        errordlg('Internal Code Error -- Invalid Data Selection Allowed')
        return
    end %switch data_selection

    case '3DM-GX3'
        %For the -GX3, analyze the appropriate data:
        switch data_selection
            case 'Euler Angles and Rates'
                %Identify which axis will be tested
                switch sensor_orientation
                    case 'Roll'
                        %Identify the initial orientation error
                        setup_roll_err = median([rpy_deg(1:tare_index,1),
rpy_rad(1:tare_index,1)]); %[deg rad]

                        if sensor_peak_time_align_checked
                            % Adjust Microstrain Sensor Time To Account for Misalignment
w/ cRIO

                            %'Fix' the sensor time by lining it up with the encoder peaks
                            [time_3DM time_error] = sensor_time_align(time_en, time_3DM,
encoder_angle, rpy_deg,sensor_orientation);
                            set(handles.text_Time_Align_Error,'Enable','on');
                        else
                            %manually adjust by the time input on the GUI
                            time_error =
str2num(get(handles.edit_Manual_Time_Adjust,'String')).*.001; %convert to milliseconds
                            time_3DM(:,1:2) = time_3DM(:,1:2) - time_error;
                            set(handles.text_Time_Align_Error,'Enable','on');
                        end

                        %interpolate the encoder data to the sensor times
                        encoder_interp(:,2) =
interp1(time_en(:,2),encoder_angle(:,2),time_3DM(:,2)); %degree, linear interp
                        encoder_interp(:,3) =
interp1(time_en(:,2),encoder_angle(:,3),time_3DM(:,2)); %rad, linear interp

                        %replace any NaN with zeros
                        encoder_interp(isnan(encoder_interp)) = 0;

                        %adjust the appropriate rpy column to account for initial
alignment errors
                        adjusted_rpy_deg = rpy_deg;
                        adjusted_rpy_rad = rpy_rad;
                        adjusted_rpy_deg(:,1) = rpy_deg(:,1) - setup_roll_err(1,1);
                        adjusted_rpy_rad(:,1) = rpy_rad(:,1) - setup_roll_err(1,2);

                        %crop the data to start at common elapsed time = 0,
                        index_pos_time = time_3DM(:,2)>=0;

```

```

        %then take the difference/calculate error
        roll_error = [time_3DM(index_pos_time,2), ...
            adjusted_rpy_deg(index_pos_time,1) -
encoder_interp(index_pos_time,2),...
            adjusted_rpy_rad(index_pos_time,1) -
encoder_interp(index_pos_time,3)];

        %update the screen data: (only count the error from after the
'still' time)
        set(handles.text_Avg_Error,'Enable','on');

set(handles.text_Avg_Error,'String',num2str(sqrt(mean(roll_error(tare_index:end,2).^2))))
;
        set(handles.text_Time_Align_Error,'String',num2str(time_error));

        %update the appdata
        setappdata(handles.Main,'Error_Roll',roll_error);
        setappdata(handles.Main,'Adjusted_rpy_deg',adjusted_rpy_deg);
        setappdata(handles.Main,'Adjusted_rpy_rad',adjusted_rpy_rad);
        setappdata(handles.Main,'Interpolated_Encoder',encoder_interp);

        %update the save variable
        Save_Vars{end+1} = 'Error_Roll';
        Save_Vars{end+1} = 'Adjusted_rpy_deg';
        Save_Vars{end+1} = 'Adjusted_rpy_rad';
        Save_Vars{end+1} = 'Interpolated_Encoder';

        case 'Pitch'
            %Identify the initial orientation error
            setup_pitch_err = median([rpy_deg(1:tare_index,2),
rpy_rad(1:tare_index,2)]); %[deg rad]

            if sensor_peak_time_align_checked
                % Adjust Microstrain Sensor Time To Account for Misalignment
w/ cRIO

                %'Fix' the sensor time by lining it up with the encoder peaks
                [time_3DM time_error] = sensor_time_align(time_en, time_3DM,
encoder_angle, rpy_deg, sensor_orientation);
                set(handles.text_Time_Align_Error,'Enable','on');
            else
                %manually adjust by the time input on the GUI
                time_error =
str2num(get(handles.edit_Manual_Time_Adjust,'String')).*.001; %convert to milliseconds
                time_3DM(:,1:2) = time_3DM(:,1:2) - time_error;
                set(handles.text_Time_Align_Error,'Enable','on');
            end

            %interpolate the encoder data to the sensor times
            encoder_interp(:,2) =
interp1(time_en(:,2),encoder_angle(:,2),time_3DM(:,2)); %degree, linear interp
            encoder_interp(:,3) =
interp1(time_en(:,2),encoder_angle(:,3),time_3DM(:,2)); %rad, linear interp

            %replace any NaN with zeros
            encoder_interp(isnan(encoder_interp)) = 0;

            %adjust the appropriate rpy column to account for initial
alignment errors

            adjusted_rpy_deg = rpy_deg;
            adjusted_rpy_rad = rpy_rad;
            adjusted_rpy_deg(:,2) = rpy_deg(:,2) - setup_pitch_err(1,1);
            adjusted_rpy_rad(:,2) = rpy_rad(:,2) - setup_pitch_err(1,2);

            %crop the data to start at common elapsed time = 0,
            index_pos_time = time_3DM(:,2)>=0;
            %then take the difference/calculate error
            pitch_error = [time_3DM(index_pos_time,2), ...
                adjusted_rpy_deg(index_pos_time,2) -
encoder_interp(index_pos_time,2),...

```

```

        adjusted_rpy_rad(index_pos_time,2) -
encoder_interp(index_pos_time,3)];

        %update the screen data: (only count the error from after the
'still' time)
        set(handles.text_Avg_Error,'Enable','on');

set(handles.text_Avg_Error,'String',num2str(sqrt(mean(pitch_error(tare_index:end,2).^2)))
);

        set(handles.text_Time_Align_Error,'String',num2str(time_error));

        %update the appdata
setappdata(handles.Main,'Error_Pitch',pitch_error);
setappdata(handles.Main,'Adjusted_rpy_deg',adjusted_rpy_deg);
setappdata(handles.Main,'Adjusted_rpy_rad',adjusted_rpy_rad);
setappdata(handles.Main,'Interpolated_Encoder',encoder_interp);

        %update the save variable
Save_Vars{end+1} = 'Error_Pitch';
Save_Vars{end+1} = 'Adjusted_rpy_deg';
Save_Vars{end+1} = 'Adjusted_rpy_rad';
Save_Vars{end+1} = 'Interpolated_Encoder';

        case 'Yaw'
            %Identify the initial orientation error
            setup_yaw_err = median([rpy_deg(1:tare_index,3),
rpy_rad(1:tare_index,3)]); %[deg rad]

            if sensor_peak_time_align_checked
                % Adjust Microstrain Sensor Time To Account for Misalignment
w/ cRIO

                %'Fix' the sensor time by lining it up with the encoder peaks
                [time_3DM time_error] = sensor_time_align(time_en, time_3DM,
encoder_angle, rpy_deg,sensor_orientation);
                set(handles.text_Time_Align_Error,'Enable','on');
            else
                %manually adjust by the time input on the GUI
                time_error =
str2num(get(handles.edit_Manual_Time_Adjust,'String')).*.001; %convert to milliseconds
                time_3DM(:,1:2) = time_3DM(:,1:2) - time_error;
                set(handles.text_Time_Align_Error,'Enable','on');
            end

            %interpolate the encoder data to the sensor times
            encoder_interp(:,2) =
interp1(time_en(:,2),encoder_angle(:,2),time_3DM(:,2)); %degree, linear interp
            encoder_interp(:,3) =
interp1(time_en(:,2),encoder_angle(:,3),time_3DM(:,2)); %rad, linear interp

            %replace any NaN with zeros
            encoder_interp(isnan(encoder_interp)) = 0;

            %adjust the appropriate rpy column to account for initial
alignment errors

            adjusted_rpy_deg = rpy_deg;
            adjusted_rpy_rad = rpy_rad;
            adjusted_rpy_deg(:,3) = rpy_deg(:,3) - setup_yaw_err(1,1);
            adjusted_rpy_rad(:,3) = rpy_rad(:,3) - setup_yaw_err(1,2);

            %crop the data to start at common elapsed time = 0,
            index_pos_time = time_3DM(:,2)>=0;
            %then take the difference/calculate error
            yaw_error = [time_3DM(index_pos_time,2), ...
                adjusted_rpy_deg(index_pos_time,3) -
encoder_interp(index_pos_time,2),...
                adjusted_rpy_rad(index_pos_time,3) -
encoder_interp(index_pos_time,3)];

```

```

        %update the screen data: (only count the error from after the
'still' time)
        set(handles.text_Avg_Error,'Enable','on');

set(handles.text_Avg_Error,'String',num2str(sqrt(mean(yaw_error(tare_index:end,2).^2))));
set(handles.text_Time_Align_Error,'String',num2str(time_error));

        %update the appdata
        setappdata(handles.Main,'Error_Yaw',yaw_error);
        setappdata(handles.Main,'Adjusted_rpy_deg',adjusted_rpy_deg);
        setappdata(handles.Main,'Adjusted_rpy_rad',adjusted_rpy_rad);
        setappdata(handles.Main,'Interpolated_Encoder',encoder_interp);

        %update the save variable
        Save_Vars{end+1} = 'Error_Yaw';
        Save_Vars{end+1} = 'Adjusted_rpy_deg';
        Save_Vars{end+1} = 'Adjusted_rpy_rad';
        Save_Vars{end+1} = 'Interpolated_Encoder';
    end %switch sensor_orientation

    % Update common appdata for this set of data:
    setappdata(handles.Main,'Sensor_rpy_deg',rpy_deg);
    setappdata(handles.Main,'Sensor_rpy_rad',rpy_rad);
    setappdata(handles.Main,'Sensor_angr',angr);
    setappdata(handles.Main,'Tare_Index',tare_index);

    %update the save variable
    Save_Vars{end+1} = 'Sensor_rpy_deg';
    Save_Vars{end+1} = 'Sensor_rpy_rad';
    Save_Vars{end+1} = 'Sensor_angr';
    Save_Vars{end+1} = 'Tare_Index';

    case 'Euler Angles Only'
        beep
        errordlg('Not Yet Configured')
        return
        %pull these
    case 'Serial Number'
        beep
        errordlg('Not Yet Configured')
        return
        %pull these
    otherwise
        errordlg('Internal Code Error -- Invalid Data Selection Allowed')
        return
    end %switch data_selection
end %case '3DM-GX3'

%% SET APPDATA FOR THE COMMON THINGS & UPDATE THE VARIABLES TO BE SAVED
setappdata(handles.Main,'Time_Encoder',time_en);
setappdata(handles.Main,'Time_Sensor',time_3DM);
setappdata(handles.Main,'Encoder_Angle',encoder_angle);
setappdata(handles.Main,'Encoder_TSPI_w',w);
setappdata(handles.Main,'Encoder_TSPI_alpha',ang_accel);
setappdata(handles.Main,'FFT_Data',[fft_en fft_w fft_alpha]);

%update the save variable
Save_Vars{end+1} = 'Time_Encoder';
Save_Vars{end+1} = 'Time_Sensor';
Save_Vars{end+1} = 'Encoder_Angle';
Save_Vars{end+1} = 'Encoder_TSPI_w';
Save_Vars{end+1} = 'Encoder_TSPI_alpha';
Save_Vars{end+1} = 'FFT_Data';

%update the save variable in the appdata
setappdata(handles.Main,'Save_Vars',Save_Vars);

%% UPDATE THE SCREEN INDICATORS TO BE "GREEN"

```

```

set(handles.text_Data_Updated,'BackgroundColor',[228/256,240/256,230/256]);
set(handles.text_Data_Updated,'ForegroundColor',[0,127/256,0]);
set(handles.text_Data_Updated,'String',{' ':'Data Updated ':''});
%Allow saving
set(handles.push_Save,'Enable','on');

% Enable the plotting Section:
set(handles.panel_Figures,'Visible','on');

% Close the msgbox that told the user it was working
if ishandle(m)~= 0
    close(m)
end

%this loads a wav file to be played if everything executes properly
finish_sound = wavread('r2d2wst1.wav');
wavplay(finish_sound,11000);

% --- Executes on button press in radio_2_Pitch.
function radio_2_Pitch_Callback(hObject, eventdata, handles)
% hObject    handle to radio_2_Pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_2_Pitch

%only allow 1 selection
set(handles.radio_2_Roll,'Value',0);
set(handles.radio_2_Pitch,'Value',1);
set(handles.radio_2_Yaw,'Value',0);

% --- Executes on button press in radio_2_Yaw.
function radio_2_Yaw_Callback(hObject, eventdata, handles)
% hObject    handle to radio_2_Yaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_2_Yaw

%only allow 1 selection
set(handles.radio_2_Roll,'Value',0);
set(handles.radio_2_Pitch,'Value',0);
set(handles.radio_2_Yaw,'Value',1);

% --- Executes on button press in radio_2_Roll.
function radio_2_Roll_Callback(hObject, eventdata, handles)
% hObject    handle to radio_2_Roll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_2_Roll

%only allow 1 selection
set(handles.radio_2_Roll,'Value',1);
set(handles.radio_2_Pitch,'Value',0);
set(handles.radio_2_Yaw,'Value',0);

% --- Executes on button press in radio_1_Pitch.
function radio_1_Pitch_Callback(hObject, eventdata, handles)
% hObject    handle to radio_1_Pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_1_Pitch

```

```

%only allow 1 selection
set(handles.radio_1_Roll,'Value',0);
set(handles.radio_1_Pitch,'Value',1);
set(handles.radio_1_Yaw,'Value',0);

% --- Executes on button press in radio_1_Yaw.
function radio_1_Yaw_Callback(hObject, eventdata, handles)
% hObject    handle to radio_1_Yaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_1_Yaw

%only allow 1 selection
set(handles.radio_1_Roll,'Value',0);
set(handles.radio_1_Pitch,'Value',0);
set(handles.radio_1_Yaw,'Value',1);

% --- Executes on button press in radio_1_Roll.
function radio_1_Roll_Callback(hObject, eventdata, handles)
% hObject    handle to radio_1_Roll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_1_Roll

%only allow 1 selection
set(handles.radio_1_Roll,'Value',1);
set(handles.radio_1_Pitch,'Value',0);
set(handles.radio_1_Yaw,'Value',0);

% --- Executes on button press in check_Plot_RPY_ALL.
function check_Plot_RPY_ALL_Callback(hObject, eventdata, handles)
% hObject    handle to check_Plot_RPY_ALL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of check_Plot_RPY_ALL

%Make this default to include the encoder
if get(handles.check_Plot_RPY_ALL,'Value') == 1
    set(handles.check_rpy_with_encoder,'Value',1);
else
    set(handles.check_rpy_with_encoder,'Value',0);
end

% --- Executes on button press in push_Plot.
function push_Plot_Callback(hObject, eventdata, handles)
% hObject    handle to push_Plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% THIS FUNCTION CREATES A SEPARATE PLOT FOR EACH ITEM CHECKED IN THE
% "OUTPUT FIGURES" PANEL

%title line to be appended to top of all title lines:
top_title = get(handles.edit_Title_Line,'String');

%Get the ylim value for the error plots:
ylim_value = str2num(get(handles.edit_ylim_value,'String'));

%% Plot 1, Encoder & Sensor, but sensor NOT adjusted for errors in intial alignment
if get(handles.check_Plot_1,'Value')==1

```

```

time_en = getappdata(handles.Main,'Time_Encoder');
time_3DM = getappdata(handles.Main,'Time_Sensor');
encoder_angle = getappdata(handles.Main,'Encoder_Angle');
figure;
hold;
xlabel('Common Elapsed Time (sec)')
title({top_title,'Encoder and Sensor Angles','(Sensor Not Adjusted For Alignment Errors)'});

%plot encoder, in rad or deg
if get(handles.radio_Select_Deg,'Value')==1
    rpy = getappdata(handles.Main,'Sensor_rpy_deg');
    ylabel('Angle (Deg)');
    plot(time_en(:,2),encoder_angle(:,2),'.')
else
    rpy = getappdata(handles.Main,'Sensor_rpy_rad');
    ylabel('Angle(Rad)');
    plot(time_en(:,2),encoder_angle(:,3),'.')
end

%plot sensor, roll/pitch/yaw, depending on selected value:
if get(handles.radio_1_Roll,'Value')==1
    plot(time_3DM(:,2),rpy(:,1),'.r')
    legend('Encoder Displacement','Sensor Roll')
elseif get(handles.radio_1_Pitch,'Value')==1
    plot(time_3DM(:,2),rpy(:,2),'.r')
    legend('Encoder Displacement','Sensor Pitch')
else
    plot(time_3DM(:,2),rpy(:,3),'.r')
    legend('Encoder Displacement','Sensor Yaw')
end
end

%% Plot 2, Encoder & Sensor, Sensor adjusted for errors in intial alignment
if get(handles.check_Plot_2,'Value')==1
    time_en = getappdata(handles.Main,'Time_Encoder');
    time_3DM = getappdata(handles.Main,'Time_Sensor');
    encoder_angle = getappdata(handles.Main,'Encoder_Angle');
    figure;
    hold;
    xlabel('Common Elapsed Time (sec)')
    title({top_title,'Encoder and Sensor Angles','(Sensor Values Adjusted For Alignment Errors)'});

    %plot encoder, in rad or deg
    if get(handles.radio_Select_Deg,'Value')==1
        rpy = getappdata(handles.Main,'Adjusted_rpy_deg');
        ylabel('Angle (Deg)');
        plot(time_en(:,2),encoder_angle(:,2),'.')
    else
        rpy = getappdata(handles.Main,'Adjusted_rpy_rad');
        ylabel('Angle(Rad)');
        plot(time_en(:,2),encoder_angle(:,3),'.')
    end

    %plot sensor, roll/pitch/yaw, depending on selected value:
    if get(handles.radio_2_Roll,'Value')==1
        plot(time_3DM(:,2),rpy(:,1),'.r')
        legend('Encoder Displacement','Adjusted Sensor Roll')
    elseif get(handles.radio_2_Pitch,'Value')==1
        plot(time_3DM(:,2),rpy(:,2),'.r')
        legend('Encoder Displacement','Adjusted Sensor Pitch')
    else
        plot(time_3DM(:,2),rpy(:,3),'.r')
        legend('Encoder Displacement','Adjusted Sensor Yaw')
    end
end

end

%% ERROR PLOTS:

```

```

%this is used to identify the points AFTER the "zero motion time," since
%we're interested in a DYNAMIC motion anyway.
tare_index = getappdata(handles.Main, 'Tare_Index');

%roll error:
if get(handles.check_Error_Plot_Roll, 'Value')==1
    time_3DM = getappdata(handles.Main, 'Time_Sensor');
    roll_error = getappdata(handles.Main, 'Error_Roll');

    figure;hold;
    if get(handles.radio_Select_Deg, 'Value')==1
        plot(roll_error(:,1),roll_error(:,2),'.b')
        plot([roll_error(1,1) roll_error(end,1)], [2 2], 'r') %top/bottom limits
        plot([roll_error(1,1) roll_error(end,1)], [-2 -2], 'r')
        ylim([-ylim_value ylim_value])
        ylabel('Roll Error (deg)');
        rms_error = sprintf('%6.3f',sqrt(mean(roll_error(tare_index:end,2).^2)));%calc
the RMS error of the dynamic part
    else
        plot(roll_error(:,1),roll_error(:,3),'.b')
        plot([roll_error(1,1) roll_error(end,1)], [2*pi/180 2*pi/180], 'r') %top/bottom
limits
        plot([roll_error(1,1) roll_error(end,1)], [-2*pi/180 -2*pi/180], 'r')
        ylim([-ylim_value ylim_value])
        ylabel('Roll Error (rad)');
        rms_error = sprintf('%6.4f',sqrt(mean(roll_error(tare_index:end,3).^2)));%calc
the RMS error of the dynamic part
    end
    xlabel('Common Elapsed Time (sec)')
    title({top_title, 'Calculated Roll Error'});
    legend(['Roll Error, (RMS = ' rms_error ')'], 'Specification Tolerance')
end

%pitch error:
if get(handles.check_Error_Plot_Pitch, 'Value')==1
    time_3DM = getappdata(handles.Main, 'Time_Sensor');
    pitch_error = getappdata(handles.Main, 'Error_Pitch');

    figure;hold;
    if get(handles.radio_Select_Deg, 'Value')==1
        plot(pitch_error(:,1),pitch_error(:,2),'.b')
        plot([pitch_error(1,1) pitch_error(end,1)], [2 2], 'r') %top/bottom limits
        plot([pitch_error(1,1) pitch_error(end,1)], [-2 -2], 'r')
        ylim([-ylim_value ylim_value])
        ylabel('Pitch Error (deg)');
        rms_error = sprintf('%6.3f',sqrt(mean(pitch_error(tare_index:end,2).^2)));%calc
the RMS error of the dynamic part
    else
        plot(pitch_error(:,1),pitch_error(:,3),'.b')
        plot([pitch_error(1,1) pitch_error(end,1)], [2*pi/180 2*pi/180], 'r') %top/bottom
limits
        plot([pitch_error(1,1) pitch_error(end,1)], [-2*pi/180 -2*pi/180], 'r')
        ylim([-ylim_value ylim_value])
        ylabel('Pitch Error (rad)');
        rms_error = sprintf('%6.4f',sqrt(mean(pitch_error(tare_index:end,3).^2)));%calc
the RMS error of the dynamic part
    end
    xlabel('Common Elapsed Time (sec)')
    title({top_title, 'Calculated Pitch Error'});
    legend(['Pitch Error, (RMS = ' rms_error ')'], 'Specification Tolerance')
end

%yaw error:
if get(handles.check_Error_Plot_Yaw, 'Value')==1
    time_3DM = getappdata(handles.Main, 'Time_Sensor');
    yaw_error = getappdata(handles.Main, 'Error_Yaw');

    figure;hold;
    if get(handles.radio_Select_Deg, 'Value')==1

```

```

        plot(yaw_error(:,1),yaw_error(:,2),'.b')
        plot([yaw_error(1,1) yaw_error(end,1)],[2 2], 'r') %top/bottom limits
        plot([yaw_error(1,1) yaw_error(end,1)],[-2 -2], 'r')
        ylim([-ylim_value ylim_value])
        ylabel('Yaw Error (deg)');
        rms_error = sprintf('%6.3f',sqrt(mean(yaw_error(tare_index:end,2).^2)));%calc the
RMS error of the dynamic part
    else
        plot(yaw_error(:,1),yaw_error(:,3),'.b')
        plot([yaw_error(1,1) yaw_error(end,1)],[2*pi/180 2*pi/180], 'r') %top/bottom
limits
        plot([yaw_error(1,1) yaw_error(end,1)],[-2*pi/180 -2*pi/180], 'r')
        ylim([-ylim_value ylim_value])
        ylabel('Yaw Error (rad)');
        rms_error = sprintf('%6.4f',sqrt(mean(yaw_error(tare_index:end,3).^2)));%calc the
RMS error of the dynamic part
    end
    xlabel('Common Elapsed Time (sec)')
    title({top_title,'Calculated Yaw Error'});
    legend(['Yaw Error, (RMS = ' rms_error ')'],'Specification Tolerance')
end

%% Raw sensor data plots:
if get(handles.check_Plot_RPY_ALL,'Value') == 1
    time_en = getappdata(handles.Main,'Time_Encoder');
    time_3DM = getappdata(handles.Main,'Time_Sensor');
    encoder_angle = getappdata(handles.Main,'Encoder_Angle');
    figure;
    hold;
    xlabel('Common Elapsed Time (sec)')
    title({top_title,'Sensor Roll, Pitch, and Yaw','(Sensor Values NOT Adjusted For
Alignment Errors)'});

    %set rpy in rad or deg
    if get(handles.radio_Select_Deg,'Value')==1
        rpy = getappdata(handles.Main,'Sensor_rpy_deg');
        ylabel('Angle (Deg)');
    else
        rpy = getappdata(handles.Main,'Sensor_rpy_rad');
        ylabel('Angle(Rad)');
    end

    %plot sensor, roll/pitch/yaw, depending on selected value:
    plot(time_3DM(:,2),rpy, '.')

    %plot encoder, if selected (in deg or rad) & then assign the legend
    if get(handles.check_rpy_with_encoder,'Value')==1 && ...
        get(handles.radio_Select_Deg,'Value')==1
        plot(time_en(:,2),encoder_angle(:,2),'.c')
        legend('Sensor Roll','Sensor Pitch','Sensor Yaw','Encoder Displacement')
    elseif get(handles.check_rpy_with_encoder,'Value')==1 && ...
        get(handles.radio_Select_Rad,'Value')==1
        plot(time_en(:,2),encoder_angle(:,3),'.c')
        legend('Sensor Roll','Sensor Pitch','Sensor Yaw','Encoder Displacement')
    else
        %no encoder to add, so put the legend in as-is:
        legend('Sensor Roll','Sensor Pitch','Sensor Yaw')
    end
end

%% Frequency Analysis Plots
if get(handles.check_Freq_Analysis,'Value')==1
    FFT_Data = getappdata(handles.Main,'FFT_Data'); %FFT[angle, w, alpha]
    Fs = str2num(get(handles.edit_Encoder_Sampling_Freq,'String'));
    num_points = length(FFT_Data);

    figure;hold

```

```

plot(linspace(-Fs/2,Fs/2,num_points),fftshift(abs(FFT_Data(:,1))));
ylabel('Magnitude of the Encoder Frequency Response');
xlabel('Frequency (Hz)');
title({top_title,'FFT of Encoder Angle'});
legend('Encoder Angle FFT')

figure;hold
plot(linspace(-Fs/2,Fs/2,num_points),fftshift(abs(FFT_Data(:,2))));
ylabel('Magnitude of the Angular Velocity Frequency Response');
xlabel('Frequency (Hz)');
title({top_title,'FFT of Calculated Encoder Angular Velocity'});
legend('Angular Velocity FFT')

figure;hold
plot(linspace(-Fs/2,Fs/2,num_points),fftshift(abs(FFT_Data(:,3))));
ylabel('Magnitude of the Angular Acceleration Frequency Response');
xlabel('Frequency (Hz)');
title({top_title,'FFT of Calculated Encoder Angular Acceleration'});
legend('Angular Acceleration FFT')

end

%% Plot Encoder Data Only , both raw and interpolated:
if get(handles.check_Plot_Encoder_Only,'Value')==1
    time_en = getappdata(handles.Main,'Time_Encoder');
    time_3DM = getappdata(handles.Main,'Time_Sensor');
    encoder_angle = getappdata(handles.Main,'Encoder_Angle');
    interpolated_encoder = getappdata(handles.Main,'Interpolated_Encoder');
    figure;
    hold;
    xlabel('Elapsed Time (sec)');
    title({top_title,'Encoder Values'});

    %set rpy in rad or deg
    if get(handles.radio_Select_Deg,'Value')==1
        plot(time_en(:,2),encoder_angle(:,2),'.');
        plot(time_3DM(:,2),interpolated_encoder(:,2),'.g');
        ylabel('Angle (Deg)');
    else
        plot(time_en(:,2),encoder_angle(:,3),'.');
        plot(time_3DM(:,2),interpolated_encoder(:,3),'.g');
        ylabel('Angle (Rad)');
    end

    xlim([0 max(time_en(:,2))])
    legend('Encoder data, raw','Encoder data, interpolated')
end

% --- Executes on button press in radio_Select_Rad.
function radio_Select_Rad_Callback(hObject, eventdata, handles)
% hObject    handle to radio_Select_Rad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_Select_Rad

%only allow 1 to be set at a time:
set(handles.radio_Select_Rad,'Value',1);
set(handles.radio_Select_Deg,'Value',0);

%update the error plot default Y-Limits:
set(handles.edit_ylim_value,'string','0.09')
set(handles.text_ylim_deg_rad,'String','rad')

% --- Executes on button press in radio_Select_Deg.
function radio_Select_Deg_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to radio_Select_Deg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_Select_Deg

%only allow 1 to be set at a time:
set(handles.radio_Select_Rad,'Value',0);
set(handles.radio_Select_Deg,'Value',1);

%update the error plot default Y-Limits:
set(handles.edit_ylim_value,'string','5')
set(handles.text_ylim_deg_rad,'String','deg')

% --- Executes on button press in push_Save.
function push_Save_Callback(hObject, eventdata, handles)
% hObject    handle to push_Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

Save_Vars = getappdata(handles.Main,'Save_Vars');

%load in all of the variables saved to the appdata
for m = 1:length(Save_Vars)
    eval([Save_Vars{m} ' = getappdata(handles.Main, '' Save_Vars{m} '');']);
end

%SET THE SAVE NAME (IN THE FUTURE THIS COULD COME FROM USER DATA OR PROMPT)
[pathstr, name, ext] = fileparts(get(handles.edit_Encoder_File,'String'));
save_name = [pathstr '\ ' name ', Analyzed Data 1.mat'];
if exist(save_name,'file') ~=0
    %allow up to 9 versions of saved data before prompting for a new str
    for k = 2:9
        if exist(save_name,'file') ~=0 %if a file exists, name it the next iteration
            save_name(end-4) = num2str(k);
        end
    end
end
end

%save all of the variables currently loaded
save(save_name,Save_Vars{:});

%begin a clean slate (regarding the save variables)
setappdata(handles.Main,'Save_Var',{ });

%only allow 1 save per analysis cycle:
set(handles.push_Save,'Enable','off');

% --- Executes on button press in check_Manual_Time_Adjust.
function check_Manual_Time_Adjust_Callback(hObject, eventdata, handles)
% hObject    handle to check_Manual_Time_Adjust (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of check_Manual_Time_Adjust

if get(handles.check_Adjust_Sensor_Times,'Value') == 1
    set(handles.check_Adjust_Sensor_Times,'Value',0);%uncheck the "Peak Align" time
adjust"
else
    set(handles.check_Adjust_Sensor_Times,'Value',1);%check the "Peak Align" time adjust"
end

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);

```

```

set(handles.text_Data_Updated,'String',{'':'Data Not Updated':''});
%disable saving changed data
set(handles.push_Save,'Enable','off');

function edit_Manual_Time_Adjust_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Manual_Time_Adjust (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Manual_Time_Adjust as text
%        str2double(get(hObject,'String')) returns contents of edit_Manual_Time_Adjust as
a double

%check the manual time adjust:
set(handles.check_Manual_Time_Adjust,'Value',1);%check the manual time adjust
set(handles.check_Adjust_Sensor_Times,'Value',0);%uncheck the "Peak Align" time adjust

%Let the user know the current GUI settings do not match what's in memory:
set(handles.text_Data_Updated,'BackgroundColor',[.961,.922,.922]);
set(handles.text_Data_Updated,'ForegroundColor',[1,0,0]);
set(handles.text_Data_Updated,'String',{'':'Data Not Updated':''});
%disable saving changed data
set(handles.push_Save,'Enable','off');

% --- Executes during object creation, after setting all properties.
function edit_Manual_Time_Adjust_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Manual_Time_Adjust (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

B. READ_ENCODER_DATA_FILE.M

```

function [time_en encoder_angle] =
Read_Encoder_Data_File(encoder_file,block_size,encoder_frame_period)
%this function reads an encoder file that has "block_size" number of
%columns for the count index, the FPGA "ticks," and the data

%The data returned are:
%time_en = [CPU time];
%encoder_angle = [CPU time, degrees, radians];

%read in the data
raw_data = textread(encoder_file,'%f'); %read as double array

%reshape the data to be in the same form as the file
raw_resaped = reshape(raw_data,3*block_size+2,length(raw_data)/(3*block_size+2));

%pull out the timestamp & index at timestamp:
time_stamp = raw_resaped(:,1);
index_at_timestamp = raw_resaped(:,2);

%separate the the index block & the encoder block
tick_block = raw_resaped(:,3:block_size+2);
index_block = raw_resaped(:,block_size+3:2*block_size+2);
encoder_block = raw_resaped(:,2*block_size+3:end);

```

```

%get sizes of new blocks:
[rowB colB] = size(index_block);

%align each into a column vector:
index = reshape(index_block',rowB*colB,1);
tick = reshape(tick_block',rowB*colB,1);
encoder = reshape(encoder_block',rowB*colB,1);

%combine them into one, and modify the encoder to be in radians & degrees:
data(:,2) = index;
data(:,3) = encoder.*360/(2^16);
data(:,4) = encoder.*2*pi/(2^16);
data(:,5) = tick.*(1/40e6); %one FPGA clock tick @ 40MHz

%using the values from index_at_timestamp, find & include the timestamp data
%%%NOTE! CURRENTLY THIS ASSUMES THAT THE VALUE @ THE TIMESTAMP WAS GOOD!
for k = 1:length(index_at_timestamp)
    time_index = data(:,2)==index_at_timestamp(k,1).*ones(length(data),1);
    data(time_index,1) = time_stamp(k,1);
end

%Next: need to rearrange the data matrix into ascending order of index
data_sorted = sortrows(data,2); %column 2 has the index

%remove all duplicate lines (use the index in col 2 to detect duplicates)
no_dupe_index = [true; data_sorted(2:end,2)~=data_sorted(1:end-1,2)];
data_no_dupe = data_sorted(no_dupe_index,:);

%% Use the existing timestamp & work backwards to timestamp all data
% use the the given encoder frame interval to timestamp each piece of data.
data_final = data_no_dupe;

%first need to identify all non-zero time indexes
non_zero_index = [find(data_no_dupe(:,1)~=0)];

%assign all the indecies a time based on all of the CPU time stamps and the
%'tick' times
data_final(:,1) = process_tick_time(data_final(:,1), data_final(:,5), non_zero_index);

%% Format the output:
time_en = data_final(:,1);
encoder_angle = data_final(:,[1 3 4 5]);

```

C. ENCODER_TSPI

```

function [en_ang,w,ang_accel,time_en,accel_xyz,g] = encoder_TSPI(data, time_en, r_pend,
steady_state,sensor_orientation)
%UPDATED 4/12/10
%NOTE: This function translates the actual encoder angle into a
%displacement angle from the original "zero" point. It assumes the avg of
%the first 'steady_state' seconds of points at rest are exactly = 0.
%This fn also takes the encoder angle and derives TSPI data for the angular
%velocity "w" and the linear acceleration depending on the orientation of
%the sensor under test. It also returns the angular acceleration "ang_accel"

%INPUTS:
% "data" should be formatted: [Time encoder(deg) encoder(rad)]
% r_pend = radius of arc, in meters
% steady_state = number of seconds at the start that the pendulum wasn't moving
% sensor_orientation = 'Roll', 'Pitch', or 'Yaw'
% -if Roll, assume Y-Axis is direction of travel, Z-Axis is radially down
% -if Pitch, assume X-Axis is direction of travel, Z-Axis is radially down
% -if Yaw, assume Z-Axis is direction of travel, x-Axis is radially up

%OUTPUTS:
% en_ang = encoder angle, with corresponding time changes

```

```

% w = angular velocity [time, deg/s, rad/s]
% ang_accel = angular acceleration [time, deg/s^2, rad/s^2]
% time_en = [Time, elapsed, delta t]
% accel_xyz = [Time_elapsed, x accel, y accel, z accel]; IN DEGREES!!!
% g = gravity used to calculate acceleration truth, based on Lat

%% Calculate delta-t
time_en(:,3) = [0; time_en(2:end,2)-time_en(1:end-1,2)];

%% Get the "zero point" & Convert COUNTS to DEGREES
% NOTE: THIS ASSUMES THE RECORDING BEGAN WITH THE PENDULUM IN THE NEUTRAL
% POSITION FOR AT LEAST X SECONDS, where X=steady_state & is input at the start.

%Identify the point where elapsed time is greater than steady_state
tare_index = find(time_en(:,2)>steady_state, 1, 'first');

%Take the most frequently occurring value (i.e. mode) in the data field
% and set this as the 'zero' point
zero_deg = mode(data(1:tare_index,2:3)); %zero_deg is formatted: [deg rad]

%Translate the encoder value into degrees of displacement at what time
en_ang(:,1) = time_en(:,2); %elapsed time
en_ang(:,2) = data(:,2) - zero_deg(1).*ones(length(data),1); %encoder angle in degrees
en_ang(:,3) = data(:,3) - zero_deg(2).*ones(length(data),1); %encoder angle in radians

%% Convert encoder position into "TRUTH" data

%Angular Velocity -- Assume Zero Initial Angular Velocity (rad/sec)
w(:,1) = time_en(:,2); %elapsed time
w(:,3) = [0; (en_ang(2:end,3) - en_ang(1:end-1,3))./time_en(2:end,3)]; %(\delta
angle)/(\delta t)
w(:,2) = w(:,3).*180/pi; %convert rad to deg

%Angular Acceleration -- Assume Zero Initial Angular Accel (rad/sec^2)
ang_accel(:,1) = time_en(:,2); %elapsed time on CPU
ang_accel(:,3) = [0; (w(2:end,2) - w(1:end-1,2))./time_en(2:end,3)]; %(\delta angular
velocity)/(\delta t)
ang_accel(:,2) = ang_accel(:,3).*180/pi; %convert rad to deg

%% If the encoder data were to be filtered, this is where you'd do it.
% % % % Filter the data and re-calculate
% % % en_ang = Encoder_Filter(en_ang,10,50,1000); %10Hz Pass, 50Hz stop, 1KHz Fs
% % % en_ang(:,1) = time_en(:,2);
% % % w(:,2) = [0; (en_ang(2:end,3) - en_ang(1:end-1,3))./time_en(2:end,3)]; %(\delta
angle)/(\delta t)
% % % w = Encoder_Filter(w,10,50,1000); %10Hz Pass, 50Hz stop, 1KHz Fs
% % % w(:,1) = time_en(:,2);
% % % ang_accel(:,2) = [0; (w(2:end,2) - w(1:end-1,2))./time_en(2:end,3)]; %(\delta
angular velocity)/(\delta t)
% % % % ang_accel = Encoder_Filter(ang_accel,10,50,1000); %10Hz Pass, 50Hz stop, 1KHz Fs
% % % ang_accel(:,1) = time_en(:,2);

%% Calculate the Linear Accelerations:
accel_xyz(:,1) = time_en(:,2);
Lat = 36.6; %Approximate Lattitude of the lab
g = 9.780318.*(1+(5.3024e-3)*(sin(Lat).^2) - (5.9e-6)*(sin(2*Lat)).^2); %gravity, from
Titterton
switch sensor_orientation
case 'Roll'
    % Assume Y-Axis is direction of travel, Z-Axis is radially down
    accel_xyz(:,3) = ang_accel(:,2).*r_pend + g * sin(en_ang(:,2)); %y axis
    accel_xyz(:,4) = (w(:,2).^2).*r_pend + g * cos(en_ang(:,2)); %z axis
case 'Pitch'
    % Assume X-Axis is direction of travel, Z-Axis is radially down
    accel_xyz(:,2) = ang_accel(:,2).*r_pend + g * sin(en_ang(:,2)); %x axis
    accel_xyz(:,4) = (w(:,2).^2).*r_pend + g * cos(en_ang(:,2)); %z axis
case 'Yaw'
    % Assume Z-Axis is direction of travel, x-Axis is radially UP

```

```

        accel_xyz(:,4) = ang_accel(:,2).*r_pend + g * sin(en_ang(:,2)); %z axis
        accel_xyz(:,2) = -(w(:,2).^2).*r_pend - g * cos(en_ang(:,2)); %x axis (note,
negative b/c it's flipped)
end

```

D. PROCESS_TICK_TIME.M

```

function time = process_tick_time(timeCPU, tick_time, non_zero_index)
%this function rectifies the time by creating a column vector of times
%based on the time the whole set was pulled. Each column contains the times
%of all, referenced to a single 'pull'. The output "time" is the average of
%all of the time vectors

%RATHER THAN USING EVERY SINGLE POINT THAT HAD A CPU TIMESTAMP, CHOOSE ONLY
%100 POINTS FOR TO GO EASY ON THE MEMORY (should still give similar results)
% all_times = zeros(length(timeCPU),length(non_zero_index)); %initialize for optimization
all_times = zeros(length(timeCPU),100); %initialize for optimization
L = length(timeCPU);
column_index = 1;

%Old version used --> 1:length(non_zero_index)
for index = round(linspace(1,length(non_zero_index),100))
    %identify the tick value of the current time
    now_tick_time = tick_time(non_zero_index(index));

    %set all_times to the current CPU time at the correct index
    all_times(non_zero_index(index),column_index) = timeCPU(non_zero_index(index));

    %identify the indicies of the times lower than current
    lower = 1:non_zero_index(index)-1;

    %identify the indicies of the times higher than current
    higher = non_zero_index(index)+1:L;

    %set the lower times (current tick time - prev)
    all_times(lower,column_index) = ...
        all_times(non_zero_index(index),column_index) - (now_tick_time -
tick_time(lower));

    %set the higher times (current tick time + next)
    all_times(higher,column_index) = ...
        all_times(non_zero_index(index),column_index) + (tick_time(higher) -
now_tick_time);

    %increment the column index
    column_index = column_index + 1;
end

%Once they're all collected Average them here!
time(:,1) = mean(all_times');
% time(:,1) = median(all_times');

```

E. SENSOR_TIME_ALIGN.M

```

function [time_3DM,time_error] = sensor_time_align(time_en, time_3DM, encoder_angle,
rpy_deg, sensor_orientation)
%this function searches blocks to try to find the sinusoidal peaks and then
%to identify the average lag between

w = 200; %this is the window size (number of points used to find encoder peak)

encoder_peak_index = []; %this will have the index of all of the encoder "peaks"

```

```

peak_time = []; %this will have the actual values of the peak time & value
was_a_peak = false;

%rpy_column is set to the col index where the roll, pitch, yaw data are,
%depending upon what the encoder was representing (matches the angles)
switch sensor_orientation
    case 'Roll'
        rpy_column = 1;
    case 'Pitch'
        rpy_column = 2;
    case 'Yaw'
        rpy_column = 3;
end

%find the encoder peaks (using a "w" point window with 10 points overlapping:
for m = 1:w-5:length(time_en)-2.*w %throw out 2 windows worth at the end
    [emax emaxI]= max(encoder_angle(m:m+w,2)); %use only degrees
    [emin eminI] = min(encoder_angle(m:m+w,2));
    %if the max index minus the min index isn't within 10 points of the
    %window size, then assume that there was a peak
    if abs(emaxI - eminI) < w-6 %throw out the variation around the end 3 pts
        %identify if it's a max or a min
        if abs(encoder_angle(m+emaxI-1,2)) > abs(encoder_angle(m+eminI-1,2)) %MAX
            encoder_peak_index = [encoder_peak_index;m+emaxI-1];
            local_peak_index = m+emaxI-1;
            was_a_peak = true;
        elseif abs(encoder_angle(m+eminI-1,2)) > abs(encoder_angle(m+emaxI-1,2))
            encoder_peak_index = [encoder_peak_index;m+eminI-1];
            local_peak_index = m+eminI-1;
            was_a_peak = true;
        end

        if was_a_peak
            %find out how many times that particular value occurs
            %consecutively, with a max number of repeated being the window size
            local_max_val_ind =
find(encoder_angle(local_peak_index:local_peak_index+w,2)...
    == encoder_angle(local_peak_index,2),w,'first');
            peak_time = [peak_time ;median(time_en(local_peak_index-
1+local_max_val_ind,2)) encoder_angle(local_peak_index,2)];

            %reset the flag
            was_a_peak = false;
        end
    end
end

%find the encoder peaks using 5 point windows and the same process:
sw = 10; %this is the sensor window size (#points used to find a peak)
sen_peak_index = []; %this will have the index of all of the encoder "peaks"
sen_peak_time = []; %this will have the actual values of the peak time & value
was_a_peak = false;
for m = 1:sw-2:length(time_3DM)-2*sw %throw out 1 window worth at the end
    [emax emaxI]= max(rpy_deg(m:m+sw,rpy_column)); %use only degrees
    [emin eminI] = min(rpy_deg(m:m+sw,rpy_column));
    %if the max index minus the min index isn't within 1 points of the
    %window size, then assume that there was a peak
    if abs(emaxI - eminI) < sw-1 %then there was a peak
        %identify if it's a max or a min
        if abs(rpy_deg(m+emaxI-1,rpy_column)) > abs(rpy_deg(m+eminI-1,rpy_column)) %MAX
            sen_peak_index = [sen_peak_index;m+emaxI-1];
            local_peak_index = m+emaxI-1;
            was_a_peak = true;
        elseif abs(rpy_deg(m+eminI-1,rpy_column)) > abs(rpy_deg(m+emaxI-1,rpy_column))
            sen_peak_index = [sen_peak_index;m+eminI-1];
            local_peak_index = m+eminI-1;
            was_a_peak = true;
        end
    end
end

```

```

        if was_a_peak
            %find out how many times that particular value occurs
            %consecutively, with a max number of repeated being the window size
            local_max_val_ind =
find(rpy_deg(local_peak_index:local_peak_index+5,rpy_column))...
            == rpy_deg(local_peak_index,rpy_column),sw,'first');
            sen_peak_time = [sen_peak_time ;median(time_3DM(local_peak_index-
1+local_max_val_ind,2)))...
            rpy_deg(local_peak_index,rpy_column)];

            %reset the flag
            was_a_peak = false;
        end
    end
end

%throw out all peaks with negative elapsed times
non_neg_e = peak_time(:,1)>0;
non_neg_s = sen_peak_time(:,1)>0;
peak_time = peak_time(non_neg_e,:);
sen_peak_time = sen_peak_time(non_neg_s,:);

% ONLY COUNT A PEAK IF IT DEVIATES MORE THAN 0.5 DEG FROM PREVIOUS PEAK
e_pk_ind = [abs(peak_time(1:end-1,2)-peak_time(2:end,2))>0.5 ; true];
s_pk_ind = [abs(sen_peak_time(1:end-1,2)-sen_peak_time(2:end,2))>0.5 ; true];
peak_time = peak_time(e_pk_ind,:);
sen_peak_time = sen_peak_time(s_pk_ind,:);

%Finally, find the difference in the peaks and take the average time delta
%and add it to the 3DM time
if length(sen_peak_time) <= length(peak_time) %then fewer sensor peaks were identified
than encoder peaks
    original_length = length(sen_peak_time);
    sen_peak_time(length(peak_time),2) = 0; %make the two the same length
    for m = 1:original_length
        [min_val en_ind] = min(abs(peak_time(m,1) - sen_peak_time(:,1))); %find the
closest match
        delta(m,1) = sen_peak_time(en_ind,1) - peak_time(m,1); %calc delta at closest
match
    end
    sen_peak_time = sen_peak_time(1:original_length,:); %reset back to original size
else %then fewer encoder peaks were identified than sensor peaks
    original_length = length(peak_time);
    peak_time(length(sen_peak_time),2) = 0; %make the two the same length
    for m = 1:original_length
        [min_val en_ind] = min(abs(peak_time(m,1) - sen_peak_time(:,1))); %find the
closest match
        delta(m,1) = sen_peak_time(en_ind,1) - peak_time(m,1); %calc
    end
    peak_time = peak_time(1:original_length,:); %reset back to original size
end

% % % %Uncomment this to plot the peaks identified, etc...
% % % figure;
% % % plot(time_3DM(:,2),rpy_deg(:,rpy_column),'m')

%DISCARD ANY TIME DELTAS OUTSIDE OF ~0.050 seconds (figure that these don't
%correspond to 'matching' peaks)
delta = delta(abs(delta(:,1)) < 0.050);

%TAKE THE MEAN AND ADD IT TO THE OUTPUT TIME.
time_3DM(:,1:2) = time_3DM(:,1:2) - mean(delta);

% % % %Uncomment this to plot the peaks identified, etc...
% % % %visualize results:
% % % hold
% % % plot(time_en(:,2), encoder_angle(:,2),'b')
% % % plot(time_en(encoder_peak_index,2),encoder_angle(encoder_peak_index,2),'r')

```

```

% % % plot(time_3DM(:,2),rpy_deg(:,rpy_column)-mean(rpy_deg(1:30,rpy_column)),'.r')
% % % plot(peak_time(:,1),peak_time(:,2),'vk')
% % % plot(sen_peak_time(:,1),sen_peak_time(:,2),'vg')
% % % disp('Done')

time_error = mean(delta);
% display(['Microstrain Sensor Timing Off By: ' num2str(mean(delta))]);

```

F. ENCODER_FILTER.M

```

function filtered_data = Encoder_Filter(data,Fpass,Fstop,Fs)
% this function takes any data input and passes the columns through an FIR
% Low pass filter defined by the given characteristics, assuming equiripple
% with a 30dB attenuation

% data = [time, any number of data columns];
% Fpass = passband frequency, in Hz;
% Fstop = stopband frequency, in Hz;
% Fs = sampling frequency, in Hz;

% filtered_data = [time, filtered data columns];

%% Determine the order of the filter
rp = .01;           % Passband ripple in dB
rs = 40;           % Stopband ripple in dB
f = [Fpass Fstop]; % Cutoff frequencies
a = [1 0];         % Desired amplitudes
% Compute deviations
dev = [(10^(rp/20)-1)/(10^(rp/20)+1) 10^(-rs/20)];
[n,fo,ao,w] = firpmord(f,a,dev,Fs);

disp(['Filter Order: n = ' num2str(n)])

%get the coefficients of the filter:
b = firpm(n,fo,ao,w);

%% Filter the Data:
time = data(:,1); %save this for later
filtered_data = filter(b,1,data); %FIR filter
filtered_data(:,1) = time(:,1);

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] IEEE Std 1554-2005, *IEEE Recommended Practice for Inertial Sensor Test Equipment, Instrumentation, Data Acquisition, and Analysis*. New York, NY: IEEE, 2005.
- [2] E. R. Bachmann, Y. Xiaoping, D. McKinney, R. B. McGhee, and M. J. Zyda, "Design and implementation of MARG sensors for 3-DOF orientation measurement of rigid bodies," *Proceedings on the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 1171–1178, September 2003.
- [3] K.-S. Chae and H.-J. Park, "Evaluation of a 3D motion sensor including accelerometer and geomagnetic sensor," *The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 900–904, August 2007.
- [4] "Haas Rotary Sale," [Online]. Available: http://www.sdecnc.com/mgxroot/page_10784.htm. [Accessed: April 12, 2010].
- [5] A. G. Cutti, A. Giovanardi, L. Rocchi, and A. Davalli, "A simple test to assess the static and dynamic accuracy of an inertial sensors system for human movement analysis," *Proceedings of the 28th IEEE Engineering in Medicine and Biology Society Annual International Conference*, pp. 5912–5915, September, 2006.
- [6] J. Shaver, "PC104 control environment development and use for testing the dynamic accuracy of the MicroStrain 3DM-GX1 sensor," Master's thesis, Naval Postgraduate School, Monterey, CA, June 2007.
- [7] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. Boston: Artech House, pp. 3–14, 21, 2008.
- [8] A. Kourepenis, J. Borentein, J. Connelly, R. Elliott, P. Ward, and M. Weinberg, "Performance of MEMS inertial sensors," *IEEE Position Location and Navigation Symposium*, pp. 1–8, April 1998.
- [9] D.H. Titterton and J. L. Weston, *Strapdown inertial navigation technology*. United Kingdom: Peter Peregrinus Ltd. on behalf of the Institution of Electrical Engineers, pp. 151–169, 39–50, 1997.
- [10] K. Maenaka, "MEMS inertial sensors and their applications," *5th International Conference on Networked Sensing Systems*, pp. 71–73, June 2008.
- [11] M. J. Thompson and D. A. Horsley, "Resonant MEMS magnetometer with capacitive read-out," *2009 IEEE Sensors*, pp. 992–995, October 2009.

- [12] X. Yun, E. R. Bachmann, H. Moore IV, and J. Calusdian, "Self-contained position tracking of human movement using small inertial/magnetic sensor modules," *2007 IEEE International Conference on Robotics and Automation*, pp. 2526–2533, April 2007.
- [13] MicroStrain, Inc., "3DM-GX1 product datasheet," [Online]. Available: <http://www.microstrain.com/pdf/3DM-GX1%20Datasheet%20Rev%201.pdf>. [Accessed: January 19, 2010].
- [14] MicroStrain, Inc., *3DM-GX1™ Data Communications Protocol*, Williston, VT: MicroStrain, Inc., March 2006.
- [15] MicroStrain, Inc., "3DM-GX3 product datasheet," [Online]. Available: http://www.microstrain.com/product_datasheets/3DM-GX3-25_datasheet_version_1.04.pdf . [Accessed: April 14, 2010].
- [16] MicroStrain, Inc., *3DM-GX3™ Data Communications Protocol: Firmware Version 0.4.14 and above*, Williston, VT: MicroStrain Inc., October 2009.
- [17] M. Um, "A position tracking system using MARG sensors," Master's thesis, Naval Postgraduate School, Monterey, CA, December 2007.
- [18] United States, Office of the Under Secretary of Defense (Comptroller), *Department of Defense Budget Fiscal Year 2011: Military Personnel Programs (M-1), Operation and Maintenance Programs (O-1), Revolving and Management Funds (RF-1)*, Washington, DC: Government Printing Office, February 2010
- [19] M. Peck, "Video games, a learning tool," *USA Today*, January 2, 2008
- [20] E. R. Bachmann, "Inertial and magnetic tracking of limb segment orientation for inserting humans into synthetic environments," PhD dissertation, Naval Postgraduate School, Monterey, CA, December 2000.
- [21] M. Tagliaferro, "Analog devices adds excitement to today's best selling digital consumer electronics products," January 4, 2007, [Online]. Available: http://www.analog.com/en/press-release/Jan_04_2007_ADI_Adds_Excitement_BestSelling/press.html, [Accessed: April 14, 2010].
- [22] Nintendo of America, "What is Wii MotionPlus?" [Online]. Available: <http://www.nintendo.com/wii/what/accessories/wiimotionplus>, [Accessed: March 25, 2010]
- [23] Apple Inc., "iPhone," [Online]. www.apple.com/iphone. [Accessed: May 5, 2010]
- [24] Gurley Precision Instruments, "Gurley model A58 absolute encoder," [Online]. Available: <http://www.gpi-encoders.com/>. p 4, [Accessed: April 22, 2010].

- [25] National Instruments, “Operating instructions and specifications NI 9403: 32 channel TTL digital input/output module,” [Online]. Available: <http://www.ni.com/pdf/manuals/374069e.pdf> [Accessed: March 25, 2010].
- [26] D. Halliday, R. Resnick, and J. Walker, *Fundamentals of Physics*, 6th Edition. New York: John Wiley & Sons, pp. 216–225, 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Chairman, Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California
4. Professor Xiaoping Yun
Naval Postgraduate School
Monterey, California
5. Professor Marcello Romano
Naval Postgraduate School
Monterey, California
6. Mr. Robert Warner
Edwards AFB, California
7. Mr. Michael Bonner
Edwards AFB, California
8. Mr. Jeremy Cookson
Edwards AFB, California